NGOP Users Guide

Version 2.1

Februrary 25, 2003

J. Fromm
K. Genser
T. Levshina
M. Mengel
V. Podstavkov

Chapter 1: Introduction

NGOP is a distributed monitoring system that provides active monitoring of software and hardware, customizable service-level reporting, early error detection, and problem prevention. NGOP provides persistent storage of collected data and is capable of executing corrective actions and sending notifications. NGOP is a framework for developing monitoring tools.

The target audience for this document is wide ranging.   It is intended that users will go directly to chapters that interest them, rather than reading the document from cover to cover.   Below is brief description of each chapter, and who should read it.

Chapter	Description	Intended Audience
1	Introduction	All users
2	Requirements	Any user installing an NGOP component.
3	Terminology	All users
4	Common Configuration Language	Users intending to write their own configuration files.
5	NGOP Central Server	Users responsible for administration of the NGOP Central Server
6	Locator Server	Users responsible for administration of the NGOP Central Server.
7 –8	Status Engine	Users responsible for administration of the NGOP Central Server. User inteding to write thier own hierarchy configuration and status rules
9	Apache/fcgi	
10	NGOP Web Monitor	Users intending to monitor components with NGOP. Users who are not interested in installing and setting up the monitor can read sections 6.1 – 6.4
11	NGOP Java Monitor	Users intending to monitor components with NGOP using standalone Java GUI.
12	Configuration File Manager	Anyone responsible for administration of the Configuration File Manager
13	Archive Server	Anyone responsible for administration of the Archive Server.
14	Monitoring Agents	Anyone that wishes to write their own monitoring agent, or is responsible for starting and stopping the agents on a machine.
15	Action Servers	Anyone responsible for writing and starting/stopping Action Servers.
16	Controlling the NGOP Daemons	Persons responsible for setting up, administrating, or installing NGOP components.

NGOP uses a centralized collection scheme. The *NGOP central server* (NCS) collects and stores information from various *monitoring agents* running on remote machines. The NCS is passive, simply listening for messages from the monitoring agents which communicate with the NCS using a well-defined protocol. NGOP provides a “plug-in” monitoring agent, which is a template that is used to easily create monitoring agents for many common tasks. A full API is also provided allowing users to create any type of monitoring agent.

Typically, a monitoring agent will monitor a piece of hardware or software and generate alarms and events to the NCS when appropriate. For example, a monitoring agent could be written to look for the presence of an important daemon and report when the daemon has died to the NCS.

The *Archive Server* is a component of NGOP that provides persistent storage. The NCS forwards all messages received from monitoring agents to the Archive Server. The Archive Server stores the messages in an Oracle database, and also provides a web based report generator as well as maintaining the database (rolling old records out to minimize the overhead for example).

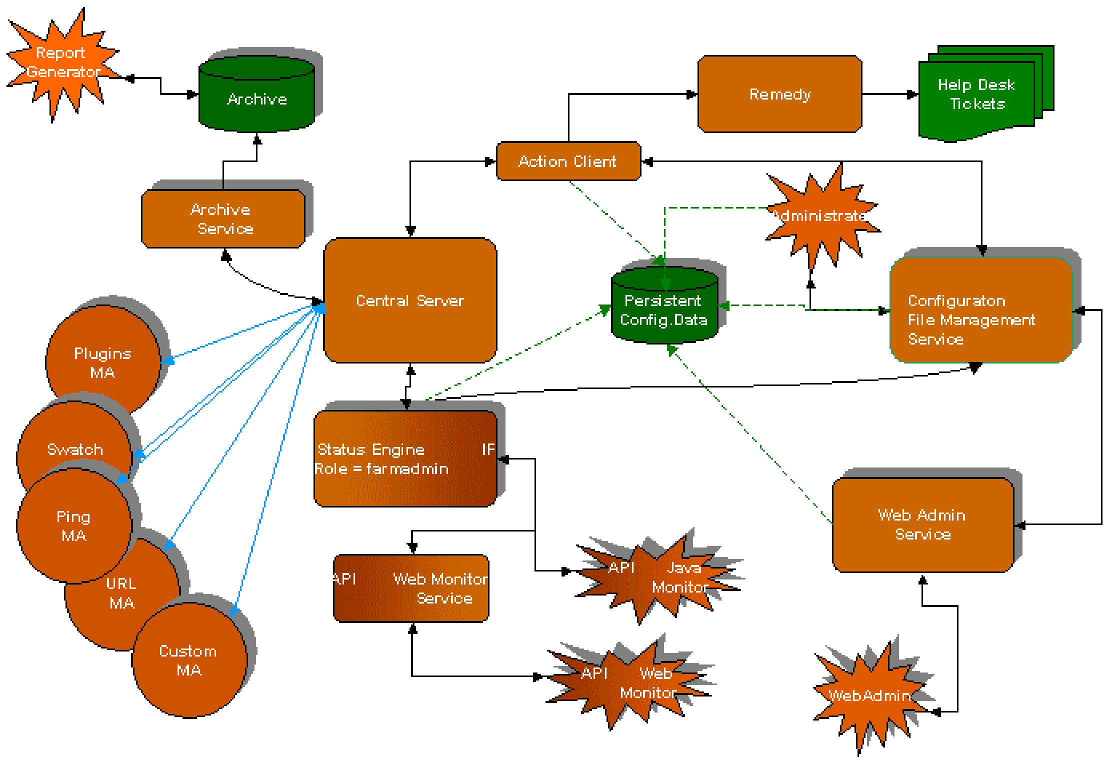
The *Configuration Server* is the component that handles all of the configuration files in NGOP. The configuration files are written using XML.

The *Status Engine* is the component that collects selected information from the NCS and processes it according to the specific rules. The Status Engine specific hierarchy configuration and rules are store in configuration files. Although the NCS is collecting information from potentially many systems, the Status Engine can susbscribe to receive date about a subset of the clusters being monitored. Multiple Status Engines can be running simulteneously each configured in such a way that refelects interested of one particular group of people (role). For example, an operations staff interested in the overall service of a system has a different view than a systems administrator who is interested in every detail. To the operations staff, having 80% of the cluster available is sufficient to provide the service, therefore they want their monitor to tell them the system is fine. The systems administrator wants to know when anything has happened in the cluster.
A full API is provided allowing users to retrieve information about a particular monitored object.

The *Location Server* is the component that registers various Status Engines and provides users with information that is used to connect with a particular Status Engine.

Information from the NGOP system is made available through the *NGOP Web Monitor* or *NGOP Java Monitor.*

Below is a pictorial view of the entire NGOP system:



See Chapter 3 for definitions of terminology.

Chapter 2: Requirements

This chapter discusses the various system requirements of various components of the NGOP system. Below is a table listing the subsystems, whether they are required for a complete NGOP system, which platforms they are available on, the requirements, and the number of instances of that subsystem that may be running. Please note that this table is a requirement for an entire NGOP system, and does not mean that each user needs to install these subsystems to begin viewing data.

Subsystem Name	Required/ Optional	Available on Platforms	Requirements	Number of Instances
NCS	Required	Linux	python 2_1 and higher fcslib v2_0 and higher	1
CFMS Broker /Indexer Admin GUI	Optional	Linux	python 2_1 built with tcl/tk support module (optional) cvs tcl v8_0_2, tkv8_0_2 (opt) blt v2_3, xed b1_0(opt)	1
Locator Server	Required	Linux	python 2_1 and higher fcslib v2_0 and higher	1
Status Engine	Required	Linux	python 2_1 and higher fcslib v2_0 and higher	N
Web Service	Requiered	Linux	python 2_1 and higher fcslib v2_0 and higher apache & f cgi imagemagick v4_0	1
Archiver	Optional	Linux, IRIX, SunOS	python 2_1 and higher dcoracle python package (available from ups as python_dcoracle) Oracle client license Shared file space for message requests.	1
Action Server	Optional	Linux	python 2_1 and higher	N
Monitoring Agents (PluginsSwatch MA Api Ping)	Required	Linux, OSF1, SunOS, IRIX	python 2_1and higher	N
Java Monitor	Optional	Linux, SunOS	Java 1.4.0	N

In order to have a complete NGOP system, a NGOP Central Server must be installed on one machine. To do anything usefull, monitoring agents will be required to monitor something, at least one status engine and monitoring web service have to be running in order to view events and alarms taking place in the system. It is possible to have more than one NGOP system, but the typical setup (here at FNAL for example) is to have one central machine that runs a site-wide NGOP.

The “Number of nodes” column refers to the number of instances that are installed to make up an NGOP system. The rows that list N simply indicate that there may be more than one of those subsystems installed in the same NGOP system. For example, there will be many monitoring agents intalled for a given NGOP system, but only one central server.

Chapter 3: NGOP Terminology

This chapter describes the terminology used when discussing the NGOP system.

3.1 Host

A Host is a computer or an entity with an assigned IP address, identified by its name.

3.2 Cluster

A Cluster is a collection of Hosts that have a common usage or purpose. Clusters may overlap. A Cluster may consist of only one Host. A Cluster is uniquely identified by its name.

3.3 Monitored Element

A Monitored Element (**ME**) is an atomic entity that is monitored by NGOP. It has a well-defined behavior, which is characterized by its state and is associated with some quantitative measurements. This entity is derived from several parts; each of them contributing to the overall State of the monitored element. A ME is located on a particular Host and belongs to a particular System. Each ME has a unique id that consists of the ME name, the Host name, the System name and the Cluster name. (Examples of Monitored Element: file system, tape drive, system daemon, and memory utilization.)

3.4 System

A System is a set of software components (ME) that are logically integrated into one unit monitored by NGOP. A System is defined on a Cluster and may be distributed across multiple Hosts. It is characterized by its State and Status. A System has a unique id that consists of the System name and the Cluster name. (Examples: LSF Batch , OS "Health" System that could contain system daemons, critical file systems, etc.)

3.5 System View

A System View is a logical collection of Systems, Monitored Elements and System Views. A System View is created by a user/administrator in order to create hierarchical structure in the NGOP Monitor. It is characterized by its Status.

3.6 Monitored Object

A Monitored Object is a System or a ME monitored by NGOP.

3.7 Monitoring Agent

A software component that monitors a particular component of the system, reporting it's status to the NGOP Central Server. Monitoring Agents are often abbreviated with MA.

3.8 State

A State is a characteristic of a Monitored Object defined by either a Monitoring Agent, or the NCS. A Monitored Object could be in four different states:

- 1 (Up) - the Monitored Object is operational
- 0 (Down) - the Monitored Object is not operational

- -1 (Undefined) - NGOP was not able to determine the state of the monitored element. This is usually set by the NCS when no information has been obtained about this object since the NCS had started.
- -2 (Unknown) - NGOP failed to determine the current state of the monitored element but was able to do it earlier. This is set by the NCS when the connection with the MA has been lost. This state indicates that at some point the NCS was communicating with the MA.

3.9 Severity Level

A Severity Level is a characteristic of an event defined by a MA. It could assume the integer values from 0 - “OK“ to 6 – “Bad”. It is used to describe events when the monitored object is still operational, but a change in the monitored object’s behavior or quantitative characteristics could indicate a potential problem. The severity level of the occurred event is redefined by the status rule in the NGOP Monitor configuration.

3.10 Event

Events are generated by MAs and describe a detected condition. An Event includes the following fields:

- System name
- Cluster name
- Monitored Element name
- Host name
- Date/Time
- Event Name (an aspect of the monitored element that contributed to event initiation).
- Event Value (the current measurements that are associated with that aspect of the monitored element).
- State
- Severity Level
- Source – the id of the Monitoring Agent (MA_name.host)
- Description (human readable explanation of the occurred event)

3.11 Status

A Status is a characteristic of a Monitored Object or System View defined by the NGOP Monitor based on the Status Rules and events. A Status of a monitored object/system view may assume the following values: “Good”, “NotInService”, “Undefined”, “Unknown”, “Warning”, “Error”, “Bad”. Status defines the color of the icons that represents system views or monitored objects in the monitor.

3.12 Status Rules

Status Rules are a set of rules defined by a user/administrator that are used by the NGOP Monitor to determine the Status of the Monitored Objects and System Views.

3.13 Action

Actions are associated with monitored objects. An event could trigger the NGOP application to send the request to the NCS to perform an action**.** NGOP generates zero or more actions depending on the event, NGOP configuration, current day/time, and requester’s authorization. Examples of Actions are:

- Display a message on the Operator console
- Send an e-mail message
- Send a message to a pager
- Run a script

Chapter 4: Common Configuration Language

This chapter describes the NGOP configuration language that allows the creation of hierarchies of monitored components, describes rules to determine the status of components, and defines when and what kind of actions should be performed. The NGOP configuration language provides a framework for creating monitoring tools ("PlugIns", "Swatch" Monitoring Agents).

The following have been defined in the NGOP configuration language:

- <For> - a looping mechanism
- <apply> - Defines a mathematical expression
- <System> - Defines and NGOP system.
- <MonitoredElement> - Defines an NGOP Monitored Element.
- <Action> - Defines an action to be taken when a condition is met.
- <If> - Defines a conditional.

Each of these are discussed in detail in the section below.

The NGOP configuration files are stored in a central repository. All NGOP configuration files are written in XML. XML stands for eXtensible Markup Language (see http://www.w3.org/XML for details). XML makes use of *tags* (words bracketed by '<' and '>') and *attributes* (of the form `name="value"`). XML uses the tags only to delimit pieces of data, and leaves the interpretation of the data completely to the application that reads it. All configuration files should conform to a corresponding DTD (Document Type Definition). A DTD is a set of rules for constructing valid XML documents.

4.1 Expansion Mechanism: `<For>` tag

The NGOP applications (such as NGOP Monitor, CFMS, PlugIns and Swatch Agent) use an expansion mechanism that allows the replication of a particular fragment of an XML document. This fragment refers to a hierarchy and is repeated for every element of this hierarchy. The hierarchy should be defined in the same XML document, or in some other XML documents referred to by name. The hierarchy consists of XML tags where each tag has at least one attribute: *Name*. There is just one outermost tag of hierarchy. This tag contains multiple tags that could be the same. This XML fragment should conform to the DTD rules.

Example:

This is a hierarchy of <Cluster> tags that contains <Host> and other <Cluster> tags . This particular example defines a cluster called "CDFFarm" which is composed of two other systems, `CDFFarmIO` and `CDFFarmWorker`. CDFFarmIO consists of the host cdffarm1, while CDFFarmWorker consists of the nodes fncdf1, fncdf2, ... , fncdf90.

```
<Cluster Name="CDFFarm">
  <Cluster Name="CDFFarmIO">
        <Host Name="cdffarm1"/>
  </Cluster>
  <Cluster Name="CDFFarmWorker">
       <Host Name="fncdf1"/>

                              .

                              .
                              .
       <Host Name="fncdf90"/>
   </Cluster>
</Cluster>
```

Each fragment of the XML document that needs to be replicated should be placed within `<For> </For>` tags. A `<For>` tag has the following attributes:

- `Each` (required) – refers to the child element within the hierarchy
- `Var` (required) – name of the variable that will be replaced every time when this name is encountered in the XML construction; `Var=”{%PlaceHolder}”`
- `In` (required) – refers to the parent element
- `Name` (required) – refers to the attribute Name of the particular parent element
- `Filename` (optional) – the name of the file where the hierarchy is described

Example:

```
<For Each=”Host” Var=”{%Host}” In=”Cluster” Name=”CDFFarm”
Filename=“CDFFarmCluster.xml “>
 <System Name=”OSHealth“  Cluster= “{%Host}” >
  <MonitoredElement Name=”ypbind” Host=”{%Host}“ Type=”Daemon“ />
  <MonitoredElement Name=”syslogd” Host=”{%Host}“ Type=”Daemon“ />
 </System>
</For>
```

The fragment of the XML document will be repeated for every `Host` tag within the `Cluster` tag with attribute `Name=”CDFFarm”`. These tags are listed in the file `CDFFarmCluster.xml`. The values of the `Cluster` attribute of a `<System>` tag and the `Host` attribute of a `<MonitoredElement>` tag will be replaced with the corresponding value of the `{%Host}` variable. The resulting configuration will look like:

```
<System Cluster=”cdffarm1” Name=”OSHealth”>
  <MonitoredElement Host=”cdffarm1” Name=”ypbind” Type=”Daemon”/>
  <MonitoredElement Host=”cdffarm1” Name=”syslogd” Type=”Daemon”/>
</System>
<System Cluster=”fncdf1” Name=”OSHealth”>
  <MonitoredElement Host=”fncdf1” Name=”ypbind” Type=”Daemon”/>
  <MonitoredElement Host=”fncdf1” Name=”syslogd” Type=”Daemon”/>
</System>
.
.
.
<System Cluster=”fncdf90” Name=”OSHealth”>
<MonitoredElement Host=”fncdf90” Name=”ypbind” Type=”Daemon”/>
<MonitoredElement Host=”fncdf90” Name=”syslogd” Type=”Daemon”/>
</System>
```

4.2 Expression `<apply>` *tag*

An `<apply>` tag defines a mathematical expression (“logical brackets” - see MathML for details). This expression is evaluated by the NGOP applications and if it is true some specific operations are carried out by the applications. For example, if at some point an expression, defined within a `<Condition>` tag in a PlugIns agent configuration file becomes true, an agent will generate an event; if an expression within a `<GenericRule>` tag becomes true, the NGOP Monitor will apply this rule to define the status of the monitored object associated with this rule.

An <apply> tag can contain other `<apply>` tags. It also could contain logical operators `(<and>, <or>, <eq>, <neq>, <lt>, <leq>,  <gt>, <geq>, <in>, <notin>.`) or functions `(<plus>, <times>,<minus>,<divide>,<sum>,<min>,<max>`). An `<apply>` element includes a `number` token element `(<cn>)` and identifier token element `(<ci>).` One of the operators or functions should be the first element within `<apply>` tag.

This XML fragment should conform to the DTD rules.

4.2.1: Examples

Example 1

Evaluate the following expression:

2y+ 4x + 1> 3z.

```
<apply>
  <gt/>
  <apply>
    <plus/>
    <apply>
      <times/>
      <ci>y</ci>
      <cn>2</cn>
    </apply>
    <!—2y--!>
    <apply>
      <times/>
      <cn>4</cn>
      <ci>x</ci>
    </apply>
    <!—4x--!>
    <cn>1</cn>
</apply>
<!—2y+4x+1--!>
  <apply>
  <times/>
  <cn>3</cn>
  <ci>z</ci>
</apply>
<!—3z--!>
</apply>
<!-- 2y+4x+1>3z -->
```

The `<sum>`,`<min>` and `<max>` tags should have the following construction:

```
<sum>
   <bvar>i</bvar>
   <lowlimit> <cn>N1</cn> </lowlimit>
   <uplimit> <cn>N2</cn>  </uplimit>
   <ci>element[i]</ci>
</sum>
```

This represents the following expression:

N2

Σ   (element[i])=element[N1]+….element[N2]

i=N1

Example 2

```
<apply>
<gt/>
 <apply>
  <sum>
       <bvar>I</bvar>
       <lowlimit> <cn>0</cn> </lowlimit>
      <uplimit> <cn>10</cn> </uplimit>
      <ci>element[i]</ci>
  </sum>
 <apply/>
 <cn>20</cn>
</apply>
```

This defines the following expression:

10

$$\sum_{i=0} (element[i])>20$$

4.3 System `<System>` tag

A <system> tag uniquely defines an NGOP system by the two tuple:
(System_Name,Cluster_Name)

A `<System>` tag indicates the beginning of the system definition and requires two attributes:
Name – defines the system name
Cluster - defines the cluster name for this system

A `<System>` tag contains multiple <MonitoredElement> tags.
This XML fragment should conform to the DTD rules.

Example:

```
<System Name="OSHealth" Cluster="Fnalu"/>
```

This defines the system "OsHealth.Fnalu". The <`For`> tag is used to define multiple systems:

```
<Unix  Name="UnixFlavor">
       <Flavor Name="Irix"/>
       <Flavor Name="Solaris"/>
       <Flavor Name="OSF1"/>
       <Flavor Name="Linux"/>
</Unix>
<For Each="Flavor" Var="{%F}" In="Unix" Name="UnixFlavor">
 <System Name="OSHealth_{%F}" Cluster="Fnalu"/>
</For>
```

The code above is equivalent to the following XML fragment:

```
<System Name="OSHealth_Irix" Cluster="Fnalu" />
<System Name="OSHealth_Solaris" Cluster="Fnalu" />
<System Name="OSHealth_OSF1" Cluster="Fnalu" />
<System Name="OSHealth_Linix" Cluster="Fnalu" />
```

4.4 Monitored Element `<MonitoredElement>` tag

A <MonitoredElement> uniquely defines an NGOP monitored element by the four tuple:
(ME_Name,Host_Name,System_Name,Cluster_Name)

A `<MonitoredElement>` tag can be encountered only within a `<System>` tag. It has the following required attributes**:**

`Name` – defines the monitored element name
`Host` – defines the physical location of monitored element. (Instances of **"**`localhost`**"** in this value will be replaced by the local host name in MA.)
`Type` – defines the type of monitored element (see Event for details)

This XML fragment should conform to the DTD rules.

Example:

```
<System Name=”OSHealth” Cluster=”Fnalu”>
  <MonitoredElement Name=”cpuLoad” Host=”fnsfo” Type=”sysUsage”/>
</System>
```

This defines the monitored element with `id=”cpuLoad.fnsfo.OSHealth.Fnalu”` and `type=”sysUsage”` .

The <`For`> tag is used to define multiple monitored elements:

```
<List Name=”Scratch”>
        <Item Name=”1”/>
        <Item Name=”2”/>
        <Item Name=”3”/>
</List>
<System Name=”OSHealth_Irix” Cluster=”Fnalu”>
  <For Each=”Item” Var=”{%I}” In=”List” Name=”Scratch”>
      <MonitoredElement Name=”/local/stage_{%I}” Host=”fnsfo” Type=”fileSystem”/>
  </For>
</System>
```

The above code fragment is equivalent to the following XML code fragment:

```
<System Name=”OSHealth_Irix” Cluster=”Fnalu” >
  <MonitoredElement Name=”/local/stage_1” Host=”fnsfo” Type=”fileSystem”/>
  <MonitoredElement Name=”/local/stage_2” Host=”fnsfo” Type=”fileSystem”/>
  <MonitoredElement Name=”/local/stage_3” Host=”fnsfo” Type=”fileSystem”/>
</System>
```

4.5 Action `<Action>` *tag*

An `<Action>` defines an action that is to be taken when a condition is met. Several optional attributes may be provided as well:

- `Method` - perform a manual or automatic action (default method is automatic)
- `Type` - execute an action locally or send request to NCS (default type is local)
- `Gap` - time (sec) before attempt to repeat the same action in case of reoccurrence of the same event
- `Counter` - the threshold that allows to generate an action if the number of ocurrences of the same event exceeded this threshold within `Gap` period
- Delay – time (sec) before attempt to perform an action, it will be executed only if condition still satisfied.

An `<Action>` contains just one other tag `<Exec>` that describes actual executable and its arguments in two required attributes:

- `Name`
- `Argument`

Special parameters are included in an *argument*; these parameters always start with % sign. Every application has a list of parameters that are used in configuration. This XML fragment should conform to the DTD rules.

Example

```
<Action ID=”email” Host=”ndem” Type=”central” Method=”automatic”>
 <Exec Name=”send_email” Argument=”%Mail,Something_awful_just_happened!” />
</Action>
```

This defines the action with `ID=”email”` that should be started automatically on the host `ndem`. The arguments that will be passed to the script `send_mail` will contain user e-mail address, and some description.

So far we have discussed the XML constructions that are common to the all NGOP subsystems, now we will concentrate on XML constructions specific to each module.

4.6 Conditional Mechanism: `<If>` tag

The <`If`> construct is used as a conditional operation in the NGOP. The only attribute is `Cond,` which specified the condition. <If>'s can be nested. An optional `<Else>` tag can be used. For the time being the value of `Cond` attribute should consist of variable placeholder `“'{%Role}'”`, logical operarator (“==”,”!=”) and role name. This XML fragment should conform to the DTD rules.

Example 1

```
<ItemList Name="CMS">
        <Item Name="CMSPROD"/>
        <If Cond="'{%Role}'=='cmsadmin'">
        <Item Name="CMSREF"/>
        </If>
</ItemList>
```

Example 2

```
<If Cond="'{%Role}'!='default'">
        <For Each="Host" Var="{%Host}" In="Cluster" Name="{%C}Worker"
                Filename="hosts_files/hostsInClusters.xml">
                .
                .
        </For>
 <Else>
        .
        .
 </Else>
</If>
```

Chapter 5: NGOP Central Server

This chapter discusses the role of the NGOP Central Server(NCS), how to start and stop it, and it's configuration.

5.1 NCS Overview

The NGOP Central Server (NCS) is a process that collects messages from multiple monitoring agents and provides clients with requested information. In particular, the NCS performs the following tasks:

- Allows for the connection of monitoring agents. The monitoring agents will send events to the NCS.
- Accepts requests from a monitoring client (Status Engine for example) to provide monitoring information.
- The monitoring client or agent can instruct the NCS to perform certain actions based on a condition. For example, Ping Agent can send the request to NCS to send email to the systems administrator if the node is failing the ping request. The NCS will not perform action iteself, the action request will be forwarded to the approriate Action Server.
- Forwards all messages sent by monitoring agents to an Archive Server.
- Once a monitoring agent has connected to the NCS, the NCS will note when the monitoring agent had died. In affect, the NCS monitors the monitoring agents.
- NCS is capable to request adminitsrative action via appropriate Action Server.

5.2 NCS Starting/Stopping

The NCS is started with the other daemons running on a host by issuing the `ups start ngop`. If UPS is not installed, then the command `ngop start` must be issued after the `$PATH` environment variable has been set to point to the NGOP directories. This command starts all of the daemons that have configuration files defined in `/var/ngop/server`.

To start only the NCS, the following command must be issued.
```
ngop start server
```
or
```
ngop start “ngop server –c /var/ngop/server/ncs.xml”
```

Conversely, to stop the NCS issue the following:
```
ngop stop server
```
or
```
ngop stop “ngop server –c /var/ngop/server/ncs.xml”
```

5.3 NCS Configuration

The NCS configuration file is written using XML. The following is a sample configuration file that is used as a template:

```
<?xml version='1.0'?>
<!DOCTYPE NCS_cfg SYSTEM "ncs.dtd">
<NCS_cfg DebugLevel="6">
   <NCS TcpPort="19996" UdpPort="19997" />
   <Client Port="7001" Host="localhost" LocalLog="log.log" Name="Archiver"/>
   <TrustedDomain>
        <Domain Name=”fnal.gov”/>
   </TrustedDomain>
   <Agent Window="5" TotalMsgNum="400" TotalMsgLength="100000" UpdateInt="2"
MissedHeartbeat="3">
   <Action ID="admin_action" Host="localhost">
            <Exec Name="do_something" Argument="arg,%Host,%ID,%Description"/>
    </Action>
  </Agent>
</NCS_cfg>
```

The NCS_cfg tag has one optional attribute that defines debug level output (0 –6) of the NCS log files. Two log files (cs.out and cs.err) are created automaticaly in ~/Log/cs directory. If directory doesn't exist it will be created. Log files are rotated daily: the old files are moved to "name.timestamp" files.
The NCS tag has two attributes, `TcpPort` and `UdpPort`. These two values must be assigned an unused port number, the NCS accepts tcp connection with various clients (e.g. status engines, action) using tcp port, and gets upd messages from all monitoring agents using udp port.
The NCS can generate request to perform action in case when a monitoring agent has died "ungracefully" or agent starts "abusing" the system by generating too many messages. The agent related information is defined by the Agent tag that has fice optional attributes. The first three attributes set the threshold for "abusive" agent definition: Window - sliding time window (minutes), TotalMsgNumber – maximum number of messages that can be generated by an agent within the sliding time window and TotalMsgLength – maximum total length of messages that can be generated by agent within the sliding time window. The last two attributes : UpdateInt defines the minumal interval between "Update" request and MissedHeartbeat defines when NCS assumes that agent is dead . In order to perform action the Action tag and Exec tag should be specified . The Action tag has two attributes ID (action id) and Host (node where Action Server is running) . The two required attributes of the Exec tag is Name (the name of the executable) and Argument (the command argument, separated by comma). The following place holder can be substituted when action is performed:
%ID – will be set to Monitoring Agent id
%Host – will be set to ip address of the node where Monitoring Agent is running
%Description – will be substitue with the following messages:
"New agent has started. NCS will ignore any messages from this Agent! Please kill it immediately!" in case if the agent with the same id has started on the same host
"Sent N messages total size L during last M min. NCS will ignore any messages from this Agent! Please kill it immediately!" - in case when agent sent too many messages during short time period.
"Monitoring Agent is dead!" in case when monitoring agent stop sending heartbeats.
The TrustedDomain tag contains list of the domains that NCS considers as trusted. Only the messages generated from the agents running on the trusted nodes will be accepted.
The `Client` tag is used to locate the host that the archive server is running on and the port that it is listening on. The final tag in the template is the `Local_log` tag, which specifies where the all messages received from the monitoring agent are to be stored. The default is to store the logging information in the file `log.log` in the ~/Log/cs. This configuration file should conform to the DTD rules.

Chapter 6: Locator Server

This chapter discusses the role of the Locator Server, how to start and stop it, and it's configuration.

6.1 Locator Server Overview

The Locator Server is the component that registers various Status Engines and assigns the unique port to each of them , so they could accept connection from various Monitoring Clients. The Locator Server provides Clients with information that is used to connect with Status Engine with a specified role .

6.2 Locator Server Starting/Stopping

The Locator Server is started with the other daemons running on a host by issuing the `ups start ngop`. If UPS is not installed, then the command `ngop start` must be issued after the `$PATH` environment variable has been set to point to the NGOP directories. This command starts all of the daemons that have configuration files defined in `/var/ngop`.

To start only the Locator Server, the following command must be issued.

```
ngop start locator
```

or

```
ngop start “ngop locator –c /var/ngop/locator/cfg.xml”
```

Conversely, to stop the Locator Server issue the following:

```
ngop stop locator
```

or

```
ngop stop “ngop locator –c /var/ngop/locator/cfg.xml”
```

6.3 Locator Server Configuration

The Locator Server configuration file is written using XML. The following is a sample configuration file that is used as a template:

```
<?xml version='1.0'?>
<!DOCTYPE LS_cfg SYSTEM "server.dtd">
<LS_cfg DebugLevel="1">
   <LS InitWait="120" MCPort="3111" SEPort="20000"/>
 </LS_cfg>
```

The ls_cfg tag has one optional attribute that defines debug level output (0 –6) of the locator server log files. Two log files (LS_cfgFile.out and LS_cfgFile.err) are created automaticaly in ~/Log/LS_cfgFile directory where cfgFile is the name of configuration file. If directory doesn't exist it will be created. Log files are rotated daily: The old files are moved to “name.timestamp” files.
The LS tag has three attributes: InitWait, `MCPort` and `SEPort`. InitWait attribute defines for how long (in seconds) the Locator Server is waiting for Status Engines to register on the Locator Server startup. During this period Locator server doesn't accept connections with Monitoring Clients. MC and SE ports must be assigned the unused port numbers, the Locator Server accepts tcp connection with various Status Engines (using SEPort) and clients (MCPort) . This configuration file should conform to the DTD rules. The Locator Server allocates the subsequent port (starting with SEPort+1) to each registered Status Engine.

Chapter 7: Status Engine

This chapter discusses the role of the Status Engine, how to start and stop it, and it's configuration and configuration of monitored heirarchy and status rules defined for monitored objects.

7.1 Status Engine

The *Status Engine* is the component that collects selected information from the NCS and processes it according to the specific rules. The Status Engine specific hierarchy configuration and rules are stored in configuration files. Although the NCS is collecting information from potentially many systems, the Status Engine can susbscribe to receive data about a subset of the systems being monitored. Multiple Status Engines can be running simulteneously each configured in such a way that refelects interested of one particular group of people (role). For example, an operations staff interested in the overall service of a system has a different view than a systems administrator who is interested in every detail. To the operations staff, having 80% of the cluster available is sufficient to provide the service, therefore they want their monitor to tell them the system is fine. The systems administrator wants to know when anything has happened in the cluster. Only one Status Engine could be running for a particular role.
A full API (see chapter) is provided allowing users to retrieve information about a particular monitored object. Web and Java Monitors are using API as well.

7.2 Status Engine Starting/Stopping

The Status Engines are started with the other daemons running on a host by issuing the `ups start ngop`. If UPS is not installed, then the command `ngop start` must be issued after the `$PATH` environment variable has been set to point to the NGOP directories. This command starts all of the daemons that have configuration files defined in `/var/ngop/status_engine`.

To start only Status Engines, the following command must be issued.
```
ngop start status_engine
```
or if one wants to start the status engine for a particular role the following command must be issued:
```
ngop start “ngop status_engine –c /var/ngop/status_engine/some_role.xml”
```

Conversely, to stop the Status Engines issue the following:
```
ngop stop status_engine
```
or
```
ngop stop “ngop status_engine –c /var/ngop/status_engine/some_role.xml”
```

7.3 Status Engine Configuration

The Status Engine configuration file is written using XML. The following is a sample configuration file that is used as a template:
```
<?xml version='1.0'?>
<!DOCTYPE status_engine_cfg SYSTEM "se.dtd">
<status_engine_cfg DebugLevel=”3”>
   <Client Port="2002" Host="ngop" Name="LSClnt"/>
   <Client Port="8080" Host="ngop" Name="CFMSClnt"/>
   <Client Port="19996" Host="ngop" Name="NCSClnt"/>
   <CfgXml CvsRep='configxml' WrkDir='.operator'
CvsRoot=':pserver:anonymous@ngop.fnal.gov:/home/ngop/Repository' Role="
operator" CfgRoot="allFermi"/>
    <TrustedDomain>
        <Domain Name=”fnal.gov”>
    </TrustedDomain>
    <CfgEvnt EventRetentionInt="24" WeekendRetentionInt="72" WeekendDay="Fri"
WeekendStartTime="17"/>
    <ColorMap>
        <Status Name="Good" Color="darkgreen" />
        <Status Name="NotInService" Color="#d2d208"/>
        <Status Name="Undefined" Color="gray"/>
        <Status Name="Unknown" Color="black"/>
        <Status Name="Warning" Color="#1670cc"/>
        <Status Name="Error" Color="orange" />
        <Status Name="Bad" Color="red" />
    </ColorMap>
    <IconMap>
        <Type Name="SystemView" Icon="systemview.gif"/>
        <Type Name="FileSystem" Icon="folder.gif"/>
        <Type Name="usrUsage" Icon="users.gif"/>
        <Type Name="sysUsage" Icon="cpuload.gif"/>
        <Type Name="memUsage" Icon="memory.gif"/>
        <Type Name="System" Icon="system.gif"/>
        <Type Name="Daemon" Icon="process.gif"/>
        <Type Name="Hardware" Icon="harddrive.gif"/>
        <Type Name="Network" Icon="network.gif"/>
        <Type Name="Fan" Icon="fan.gif"/>
        <Type Name="Temperature" Icon="temperature.gif"/>
        <Type Name="Processor" Icon="multiproc.gif"/>
        <Type Name="MonitoredElement" Icon="blank.gif"/>
        <Type Name="webpage" Icon="html.gif"/>
    </IconMap>
</status_engine_cfg>
```

The status_engine_cfg tag has one optional attribute that defines debug level output (0 –6) of the status engine log files. Two log files (StatusEngine_cfgname.out and StatusEngine_cfgname.err) are created automaticaly in ~/Log/StatusEngine_cfgname directory, where “cfgname” is the name of configuration file. If directory doesn’t exist it will be created. Log files are rotated daily: the old files are moved to “name.timestamp” files.
Status Engine is established permenent tcp connections with the Locator Server, the NCS and optionaly with the CFMS. The port and host of the corresponding daemon process are specified with the tag “Client” where attribute “Name” should have corresponding value : LSClnt, NCSClnt or CFMSClnt.

The CfgXml tag defined the location of configuration,status rules and default files (attribute WrkDir) , the cvs repository and root names (CvsRepository and CvsRoot) , status engine role (“Role”) and root of the configuration hierarchy (CfgRoot). The cvsRoot attribute should be specified if the configuration should be downloaded via CFMS from cvs, and cfgRoot should be specified if the hierarchy root has to be changed.
The TrustedDomain tag contains list of the domains that Status Engine considers as trusted. It handles pending action only if request to execute/cancel it came from trusted node.
 In order to statrt Status engine without connecting to CFMS, you have to placed all the hierarchy configuration , status rule and default files (see ...) under the directory wrkDir/cvsRepository.
The CfgEvnt tag defines the storage parameters of all the events, alarm and actions. This tag is optional as well as all its attributes. The attributes are defined the following:

EventRetentionInt – duration while all the unacknowledged events,alarms and actions will be stored during weekdays (hours)

WeekendRetentionInt - duration while all the unacknowledged events,alarms and actions will be stored during weekends (hours)

WeekendDay – day of the week (“Fri”) when weekend starts

WeekendStartTime – time of the day (hour) when weekend starts

ColorMap and IconMap define available object statuses and types, and provide the mapping between Statuses and Colors as well as monitored object Types and Icons. This configuration file should conform to the DTD rules.

7.4 Default Configuration Files

There are several configuration files that contain general information needed for the NGOP Status Engine. These files include data about “out of service” monitored objects, available service classes, existing hosts and clusters.

These files will be downloaded into specified configuration area. These are considered the default configuration files. These files also should be copied into your local area should you choose to create your own custom configuration. Templates of these files can be found in the directory $NGOP_DIR/templates/central_configuration/. The name of these files can be anything, but certain naming conventions have evolved. Common configuration files are discussed next.

7.4.1 File service_class.xml

The service_class.xml configuration file contains information about defined types of service. The service type is associated with the hosts and monitored objects. By default, a monitored element, located on a host has the same service type as this host. A service type defines the time period of active monitoring.

This file has the following required declarations and tags:

```
<?xml version='1.0'?>
<!DOCTYPE NGOPConfig SYSTEM "service_class.dtd">
<NGOPConfig>
<Default_File/>
<ServiceClass>

……… - definition of service type should be placed here
 </ServiceClass>
</NGOPConfig>
```

A `<ServiceClass>` tag contains definition of the several service types (tag `<ServiceType>` , such as "`8to17by5`" or "`24by7".` The default service type is "`24by7`".

A service type is described by a mathematical expression by using an `<apply>` tag. If the expression is evaluated to be false, all events occurred with the corresponding monitored object/host will be ignored. Within an `<apply>` tag, a `<ci>` tag could assume only two values: `hour` or `day_of_the_week`. Days of the week are represented by an array of integers, where 0 corresponds to Monday. `Hour` is represented by an integer value within 0 – 24 range. This configuration file should conform the to the DTD rules.

Example:

```
<ServiceType name="8to17by5">
  <apply>
       <and/>
       <apply>
       <geq/>
             <ci>hour</ci>
             <cn>8</cn>
       </apply>
       <!—(hour>=8)--!>
       <apply>
              <leq/>
              <ci>hour</ci>
              <cn>17</cn>
       </apply>
       <!—(hour<=17)--!>
       <apply>
       <notin/>
       <ci>day_of_the_week</ci>
       <cn>[5,6]</cn>
       </apply>
       <!—(day_of_the_week not in [Saturday,Sunday])--!>
  </apply>
<!—this just means that "8to17by5" service type is defined between 8:00-17:00 every day
except Saturday and Sunday--!>
<!—see apply for details--!>
</ServiceType>
```

7.4.2 File `hosts_in_clusters.xml`

The `hosts_in_clusters.xml` configuration file contains clusters and hosts that exist in the system. The service type of each host is defined in this configuration. If a service type is not defined, the default service type is assumed for a host. This file has the following required declaration and tags:

```
<?xml version='1.0'?>
<!DOCTYPE NGOPConfig SYSTEM "hosts_in_clusters.dtd">
<NGOPConfig>
<Default_File/>
<HostsInClusters>

……… - known status definition should be placed here
 </HostsInClusters>
</NGOPConfig>
```

A `<HostsInClusters>` tag contains multiple <`Cluster`> tags. A <`Cluster`> tag has one required attribute (`Name`).

A `<Cluster>` tag contains other `<Cluster>` or `<Host>` tags. A `<Host>` tag also has `Name` as the only required attribute.

A <`ServiceType`> tag is placed anywhere within a <`HostsInClusters`> tag. It is defined the service type for all clusters and hosts it contains. A `<ServiceType>` tag has <`Name`> as the one required attribute. `Name` contains the name of the service type defined in `service_class.xml`.
<Default_File> DTD
This configuration file should conform to the DTD rules.

Example:

```
<ServiceType Name="24by7">
<Cluster Name="FNALU_BATCH">
  <Cluster Name=" FNALU_BATCH_OSF1">
       <Host Name="fdei01"/>
  </Cluster>
  <Cluster Name=" FNALU_BATCH_IRIX">
       <Host Name="fsgb02"/>
        <Host Name="fsgb03"/>
        <Host Name="fsgi02"/>
        <Host Name="fsgi03"/>
  </Cluster>
  <Cluster Name=" FNALU_BATCH_Solaris">
        <Host Name="fsub01"/>
        <Host Name="fsui02"/>
        <Host Name="fsui03"/>
  </Cluster>
</Cluster>
</ServiceType>
```

This example describes the cluster `FNALU_BATCH`. It has three sub clusters:

- `FNALU_BATCH_IRIX` with hosts:
    - `fsgb02`
    - `fsgb03`
    - `fsgi02`
    - `fsgi03`
- `FNALU_BATCH_OSF1` with host:
    - `fdei01`
- `FNALU_BATCH_Solaris` with hosts:
    - `fsub01`
    - `fsui02`

   o fsui03

All hosts that belong to the FNALU_BATCH cluster require 24by7 maintenance support.

7.4.3 File kn_st.xml

The kn_st.xml (known status) configuration file contains references to the monitored objects or hosts that are known to be out of service for a significant period of time. A monitored object/host is marked as "bad", "in repair" or "test". If a monitored object/host is not listed in this file, its status is working. This file has the following required declaration and tags:

```
<?xml version='1.0'?>
<!DOCTYPE NGOPConfig SYSTEM "known_status.dtd">
<NGOPConfig>
<KnownStatus>

……… - known status definition should be placed here
 </KnownStatus>
</NGOPConfig>
```

A <KnownStatus> tag contains multiple <Status> tags. A <Status> tag has one required attribute; Name, that can assume the values "bad", "in_repair", or "test".

You can specify the "out of service" time interval (<OutOfServiceInterval> tag) within the <Status> tag. It includes one optional attributes Description,User and the following required attributes:

    StartDateTime – "yyyy-mm-dd hh:mm"
    EndDateTime – "yyyy-mm-dd hh:mm"

Out of service monitored objects and hosts are listed within the corresponding <Status> tag. This configuration file should conform to the DTD rules.

Examples:

```
<Status Name="bad">
  <Host Name="fnpc110"/>
  <System Name="LSF" Cluster="fsgb02" />
</Status>
```

This declares host fnpc110 and system LSF.fsgb02 to be in a known bad condition.

```
<Status Name="in_repair">
  <OutOfServiceInterval StartDateTime="2001-05-01 12:30">
   <System Name="OCS" Cluster="FixTarget"/>
  </OutOfServiceInterval>
</Status>
<Status Name="test">
  <OutOfServiceInterval StartDateTime="2001-05-04 08:30">
   <System Name="FBS" Cluster="MovingTarget"/>
  </OutOfServiceInterval>
</Status>
```

This declares the system OCS.FixTarget to be in repair since May 1, 2001 12:30 and host "fnpc107" being used for testing purpose weekly from 8 am to 12 pm since May 4, 2001

7.5 NGOP Hierarchy Definition

An NGOP monitored hierarchy consists of system views, systems, and monitored elements. The system and system view definitions are placed in one or multiple configuration files. The monitored element definitions should be always placed within the system definition. Every configuration file describing the NGOP monitored hierarchy has the following required declaration and tags:

```
<?xml version='1.0'?>
<!DOCTYPE NGOPConfig SYSTEM "hierarchy.dtd">
<NGOPConfig>

……… - definition of system view, system, and monitored elements should be placed here
</NGOPConfig>
```

The following XML tags are used to describe the monitored hierarchy:

`<SystemView>`
`<System>`
`<Monitoried Element>`

A <`For`> tag can be used anywhere in the monitored hierarchy definition in order to replicate some XML fragments.

7.5.1 System View

A System View is uniquely defined by its id. A system view contains only references to the other system views and monitored objects. (**Important**: all components of the hierarchy should be defined elsewhere!)

A <`SystemView`> tag has the following attributes:

`ID` (required)
`RefRule` - a reference to the status rule set, describing the status rules for this system view, the default value is "SystemViewDefRuleSet"

This configuration file should conform to the DTD rules.

Example 1:

```
<SytemView ID="LSF_Fnalu_Batch">
    <SystemView ID="Fnalu_Batch_Irix"/>
    <SystemView ID="Fnalu_Batch_Solaris">
    <!—references to the system views---!>
    <System Name=Ping Cluster="Fnalu_Batch"/>
    <!—reference to the system--!>
    <System Name="OSHealth" Cluster="Fnalu_Batch">
       <MonitoredElement Name="/tmp" Host= Host="fsgb02"/>
       <MonitoredElement Name="/tmp" Host="fsgb03" />
        ….
    </System>
    <!—references to monitored elements--!>
</SystemView>
```

This example defines a system view `LSF_Fnalu_Batch` that contains two other system views (`Fnalu_Batch_Solaris` and `Fnalu_Batch_Irix`), one system (`Ping.Fnalu_Batch`), and several monitored elements `(/tmp/fsgb03.OSHealth.Fnalu_Batch` for example).

Example 2:

The following example defines system views `Fnalu_Batch_Irix` that contains three LSF systems running on nodes named `fsgb02, fsgb03`, and `fsgi02`.

```
<SystemView ID="Fnalu_Batch_Irix">
         <System Name="LSF" Cluster="fsgb02"/>
         <System Name="LSF" Cluster="fsgb03"/>
         <System Name="LSF" Cluster="fsgi02"/>
</SystemView>
```

7.5.2 System

A <`System`> tag contains multiple <`MonitoredElement`> tags and should be referenced at least once within <SystemView> tag. A definition of a system hierarchy should be placed outside system view scope. In the NGOP hierarchy definition a <System> tag has two additional optional attributes:

`ServiceType` – default "`24by7`"

`RefRule` - a reference to the status rules set, describing the status rules for this system, the default value is "SystemDefRuleSet"

This configuration file should conform the DTD rules.

Example:

The following example defines a system called `OSHealth.Fnalu` that is monitored around on a 24by7 basis. The status rule set defining the status of this system is described in `SGIHealthRuleSet`. The system consists of several monitored elements ("ping.fsgb02.Ping.Fnalu_batch" for example).

```
<System Name="Ping" Cluster="Fnalu_Batch" ServiceType="24by7"
        RefRule="SGIHealthRuleSet">
        <MonitoredElement Name="ping" Host="fsgb02" Type="Hardware"/>
        <MonitoredElement Name="ping" Host="fsgb03" Type="Hardware"/>
        <MonitoredElement Name="ping" Host="fsub02" Type="Hardware"/>
</System>
```

7.5.3 Monitored Element

A <`Monitored Element`> tag is encountered only within <`System`> tags and has two additional optional attributes:

`ServiceType` – default is service type of the host

`RefRule` – a reference to the status rule set, describing the status rules for this monitored element, the default value is "MEDefRuleSet"

This configuration file should conform to the DTD rules.

Example:

The following example defines the monitored elements with an `id` of `cpuLoad.fnsfo.OSHealth.Fnalu` and a `Type` of `sysUsage`. The status rule set defining the status of this monitored element is described in `MEDefRuleSet` and the service type is the service type of the host `fnsfo`.

```
<System Name="OSHealth" Cluster="Fnalu">
        <MonitoredElement Name="cpuLoad" Host="fnsfo" Type="sysUsage"/>
</System>
```

7.6 Status Rule Sets

Every set of status rules is associated with some systems view or monitored objects. When the NGOP Monitor receives an event regarding an object, it uses set of status rules associated with this object to define its status and severity level. It also applies the corresponding rules to every component of the hierarchy to which this object belongs. In the NGOP configuration, a `<StatusRuleSet>` tag with required attribute ID represents the set of status rules. Every set of status rules definition is located in a separate file and has the following required declaration and tags:

```
<?xml version='1.0'?>
<!DOCTYPE NGOPRules  SYSTEM "rules.dtd">
<NGOPRules>
<StatusRuleSet ID="MEDefRuleSet">

……dependent list could be placed here
…… rules
```

```
</StatusRuleSet>
</NGOPRules>
```

The content of the set of status rules definition is divided into two parts:

- Dependent list - list of all objects that this particular monitored object depends on
- Rules

A Dependent list is omitted if a monitored object doesn't depend on any other object. This configuration file should conform to the DTD rules.

7.6.1 Dependent List

A dependent list contains a list of the references to monitored objects and system views. In the NGOP configuration, a `<DependList>` tag represents a dependent list. In a dependent list, monitored objects/system views are arranged in groups. A group may contain other groups and is represented by a `<Group>` tag that has one required attribute `Name` (it should be unique only within this `<SatusRulesSet>` definition). Every group has a parameter `"%GroupLen"` that is equal to the total number of monitored objects in the group. A system may contain one special empty group with the attribute `Name` set to `"{self}"`. It means that this system depends on all monitored elements that it contains. All objects in a dependent list are ordered by their appearance relative to a particular group.   A `<For>` tag may be used in a dependent list. This XML fragment should conform to the DTD rules.

Example:
This is an example of dependent list that consist of the `"self"` group:

```
<DependList>
<Group Name = "{self}"/>
</DependList>
```

The FBS system is a batch system developed at Fermilab. FBS depends on a `bmgr` and `logd` process running on a central node. FBS depends on the central node being up.   FBS also depends on a process called a *launcher* to be running on all nodes in the system that can run a batch process. FBS runs on a cluster. In this example, the clusters `CDFFarm` and `D0Farm` (defined in the `HostsInClusters.xml` file) are running the FBS system.

`HostsInClusters.xml:`

```
<?xml version='1.0'?>
<!DOCTYPE NGOPConfig SYSTEM "ngop_default.dtd">
<NGOPConfig>
<Default_File/>
 <HostsInClusters>
 <Cluster Name="CDFFarm">
  <Cluster Name="CDFFarmIO">
       <Host Name="cdffarm1"/>
  </Cluster>
  <Cluster Name="CDFFarmWorker">
       <Host Name="fncdf1"/>
       ….
       <Host Name="fncdf90"/>
  </Cluster>
 </Cluster>
 <Cluster Name="D0Farm">
   <Cluster Name="D0FarmIO">
       <Host Name="d0bbin"/>
   </Cluster>
   <Cluster Name="D0FarmWorker">
        <Host Name="fnd01"/>
              ….
        <Host Name="fnd100"/>
    </Cluster>
 </Cluster>
```

```
….
 </HostsInClusters>
</NGOPConfig>



<?xml version='1.0'?>
<!DOCTYPE NGOPRules  SYSTEM “rules.dtd”>
<NGOPRules>
<FBSInstance Name=”FBS”>
   <Instance Name=”D0”/>
   <Instance Name=”CDFFarm”/>
</FBSInstance>
<For Each=”Instance” Var=”{%I}” In=”FBSInstance” Name=”FBS”>
  <StatusRuleSet ID=”FBS{%I}RuleSet>
  <DependList>
    <Group Name=”fbs_daemon/>
      <System ID=”FBS” Cluster=”{%I}Farm” >
     <For Each=”Host” Var=”{%H}” In=”Cluster” Name=”{%I}FarmIO”
              Filename=”HostsInClusters.xml”>
        <MonitoredElement Name=”bmgr” Host=”{%H}” />
        <MonitoredElement Name=”logd” Host=”{%H}” />
     </For>
   </System>
  </Group>
<!—logd could be referenced in DependRule as fbs_daemon[1]--!>
<Group Name=”launcher”>
  <System ID=”FBS” Cluster=”{%I}Farm” >
   <For Each=”Host” Var=”{%H}” In=”Cluster” Name=”{%I}FarmWorker”>
        <MonitoredElement Name=”launcher Host=”{%H}”/>
   </For>
   </System>
</Group>
<!—launcher on fncdf1 could be referenced in DependRule as launcher[0]--!>
<Group Name=”hostUP”/>
  <System Name=”Ping” Cluster=”{%I}FarmIO”>
    <For Each=”Host” Var=”{%H}” In=”Cluster” Name=”{%I}FarmIO”>
        <MonitoredElement Name=”ping” Host=”{%H}”/>
    </For>
  </System>
</Group>
</DependList>

…
</StatusRuleSet>
</NGOPRules>
```

7.6.2 Rule

When the NGOP Monitor receives an event it performs the following steps:

1. Finds the monitored object associated with this event
2. Finds the status rule set that defined rules for this monitored object
3. Evaluates an expression defined in every rule
4. Applies the rule (sets status and severity level) if an evaluated expression is true.   The worst status/severity level of the corresponding rule with the highest priority will determine ultimate object’s status/severity level.
5. Identifies all the members of the hierarchy that are affected by the change of this monitored object status.
6. Repeats steps 2-6 until there are no more affected members of hierarchy (step 5).

There are two implemented rule types.

- A Generic Rule (<`GenericRule`> tag) sets the monitored object status and severity level based on the event received from the NCS.

- A Dependent Rule (<`DependRule`> tag) sets the monitored element status and severity level based on the event received from the NCS and the status of each dependent monitored object in some group.

All these rules have three required attributes:

- `Status` – This can assume a special value "`None`" indicates that this rule will not change an existing status. In a dependant rule the `Status` of dependent list members is used in the expression.
- `Prio (Priority)` – This indicates the importance of the particular rule. It can assume any integer value greater than or equal to 0. The lower the value, the less important the rule is. If several rules are satisfied, the status and severity level of the monitored object will be the one associated with the rule with the highest priority.
- `SevLevel (Severity Level)` – This can assume a special value of "`None`" that indicates that this rule will not change the existing severity level.

There is one optional attribute:

- `Dsc (Description)`. – Description is an explanation of the condition of a rule. Special parameters may be included in a description such as `%ID` , `%Host` or `%Event`. These parameters will be replaced by the corresponding values of the monitored object associated with this rule.

Every rule contains an expression that has to be evaluated upon receipt of an event. In an expression any particular field of the event is referred by its name. An Action can be attached to any of the rules.

Example:

Let's assume that the agent "`LinuxHealth`" is monitoring the file system `"/export/home"` on the worker node `"fnd01"`. This file system should be mounted from the I/O node `"d0bbin"`. The `LinuxHealth` Agent can generate events in three cases:

1. The file system is not mounted
2. Automount program is not running
3. The file system is more than 95% full

The status of the monitored element should change upon receiving any of these events unless the I/O node is down.

In order to do so the set of status rules (`FileSystemRuleSet`) should include the following:

Rule Type	Status	Priority	Evaluated Expression
Dependent	Good	1	d0bbin is down
Generic	Bad	0	File system is not mounted
Generic	Error	0	Automount is not running
Generic	Warning	0	File system is 95% full

If at some point we receive event 1 (The file system is not mounted), the status becomes "Bad" if the I/O node is up and "Good" if the I/O node is down. The monitoring of the status of the I/O node should be done from another location. In this way, the failure of the I/O node will not affect the agent monitoring it.

7.6.3 Generic Rule Example

This rule is applied to a particular monitored object if the event associated with this object has a `"State"` of 1 ("Up"). The severity level remains unchanged. This XML fragment should conform to the DTD rules.

```
<GenericRule Status="Good" Prio="0" SevLevel="None" >
```

```
<apply>
<eq/>
        <ci> State </ci>
        <cn> 1 </cn>
</apply>
<!—if expression (State==1) is true , rule is applied--!>
</GenericRule>
```

This rule is applied to a particular monitored object if the event associated with this object has `“State”` value equal to `0` (“Down”). The severity level remains unchanged.

```
<GenericRule Status=”BAD” Prio=”0”  SevLevel=”None” >
<apply>
        <eq/>
        <ci>State</ci>
        <cn>0</cn>
</apply>
<!-- if exression (State == 0) is true then the rule is applied -->
</GenericRule>
```

This rule is applied to a particular monitored object if the event associated with this object has `“State”` value equal to 1 (“Up”) and `“SevLevel”` value equal to 6 (“Bad”). It set status to `“Error”`.

```
<GenericRule Status=”Error” Prio=”0” SevLevel=”None”>
<apply>
<and/>
<apply>
        <eq/>
        <ci>State</ci>
        <cn>1</cn>
</apply>
<apply>
        <eq/>
        <ci>SevLevel</ci>
        <cn>6</cn>
</apply>
</apply>
<!—if expression ((State==1) && (SevLevel==6)) is true , rule is applied--!>
```

7.6.4 Dependent Rule

A Dependent Rule allows for the use of objects from a dependent list in an expression. These objects are indexed by their position within a specific group of a dependent list. For example, the object that is listed third in the group named `“fbs_daemon”` is referred as `“fbs_deamon[2]”` (indexing starts with 0) in an expression. If a dependent list of a system status rule set has a group with `Name=”{self}”`, the i-th monitored object that belongs to this system is referred as `“{self[i-1]}”`. This XML fragment should conform to the DTD rules.

Example

This rule is applied to the “`FBS`” system when NGOP reports that the `bmgr` daemon is not running. `bmgr` is the first element (`fbs_daemon[0]`) of the `fbs_daemon` group in the dependent list of the `FBS` rule (see dependent list Example)

```
<DependRule Status=”Bad” Prio=”1”  SevLevel=”None” Dsc=”Batch_Manager_is_down”>
<apply>
<and/>
<apply>
        <eq/>
        <ci>fbs_daemon[0].EventType</ci>
        <cn>”Daemon”</cn>
</apply>
<apply>
        <eq/>
        <ci>fbs_deamon[0].State</ci>
        <cn>0</cn>
</apply>
```

```
</apply>
<!—if expression ((bmgr.EventType==”Daemon”) && (bmgr.State==0)) is true then  rule is
applied--!>
```

This rule is applied to the `FBS` system when NGOP reports that the `FBS` central machine is down. `ping` is the first element of the “hostUp” group (`hostUp[0]`)in the dependent list of the `FBS` rule (see dependent list Example).

```
<DependRule Status=”Unknown” Prio=”1” Dsc=”%Host_is_down”>
<apply>
        <eq/>
        <ci>hostUp[0].State</ci>
        <cn>0</cn>
</apply>
</DependRule>
```

Chapter 8: Status Engine API

The Status Engine API provides access to Status Engine run-time and configuration information about a particular monitored object. The API front-end class SEClient class performs communication between an API client (e.g Web Monitor) application and Status Engine.
In order to use Status EngineAPI, user application must create an object of this class. An SEClient object provides methods for:

- Obtaining the information about monitored object status, state, severity level, type etc
- Obtaining information about monitored object heirarchy
- Obtaining information about events, alarms and actions associated with a particular monitored object
- Acknowledged chosen events, alarms and actions associated with a particular monitored
- Initiate update request
- Initiate performance of manual pending actions

Python and Java binding are available to this time. This chapter describes the SEClient class of the Status Engine Python API.

8.1 Constructor SEClient

Purpose: creates new SEClient object. Creates a connection to the Status Engine for a particular user/role.

Synopsis:

- `SEClient(role,user,port,host)`

Arguments:

- role - Status Engine Role
- user - Unix name of the client
- port – Locator Server port
- host – Locator Server host

Return value:

- SEClient object

Example:

```
se = SEClient(“operator”,”smith”,5001,”apple.fnal.gov”)
```

8.2 bye()

Purpose: Gracefully disconnects from Status Engine

Synopsis:

- `bye()`

Arguments:

- None

Return value:

- None

Example:
```
se.bye()
```

8.3 TreeGetRoot()

Purpose: Return the root of the hiearchy tree.
Synopsis:

- `TreeGetRoot()`

Arguments:

- None

Return value:

- String - root of hierarchy tree

Example:
```
rid = se.TreeGetRoot()
```

8.4 TreeGetKids()

Purpose: Return a list of children ids rooted by specified parent id
Synopsis:

- `TreeGetKids(rid)`

Arguments:

- rid – parent id

Return value:

- list of Stings – children ids

Example:
```
/* Return all children */
rid = se.TreeGetRoot()
kids = se.TreeGetKids(rid)
```

8.5 GetLastHeard()

Purpose: last update recieved from the NCS
Synopsis:

- GetLastHeard()

Arguments:

- None

Return value:

- Float – Unix time : last update received from the NCS

Example:
tm=se.GetLastHeard()

7.6 GetUpdateRequest()

Purpose: initiates update request fro a particular monitored object. It could be requested only for System or Monitored Element.
Synopsis:
- GetUpdateRequest(id)

Arguments:
- String – id of monitored object

Return value:
- None

Example:
se.GetUpdateRequest(rid)

8.7 GetStatus()

Purpose: Obtains status of a particular monitored object.
Synopsis:
- GetStatus(id)

Arguments:
- String – id of a particular monitored object

Return value:
- String – status (e.g. Bad, Good, Error…)

Example:
status=se.GetStatus(id)

8.8 GetKnownStatus()

Purpose: Obtains "known" status of a particular monitored object.
Synopsis:
- GetKnownStatus(id)

Arguments:
- String – id of a particular monitored object

Return value:
- String – "known" status (e.g. working, test, in repair…)

Example:
knownStatus=se.GetKnownStatus(id)

8.9 GetServiceType()

Purpose: Obtains service type of a particular monitored object.
Synopsis:
- GetServiceType(id)

Arguments:
- String – id of a particular monitored object

Return value:

- String – service type (e.g. 24by7,8to17by5 etc)

Example:
st=se.GetServiceType(id)

8.10 GetSevLevel()

Purpose: Obtains severity level of a particular monitored object.
Synopsis:

- GetSevLevel(id)

Arguments:

- String – id of a particular monitored object

Return value:

- String – severity level (e.g. Warning, Error,NotInService)

Example:
sl=se.GetSevLevel(id)

8.11 GetState()

Purpose: Obtains state of a particular monitored object.
Synopsis:

- GetState(id)

Arguments:

- String – id of a particular monitored object

Return value:

- Int – state (e.g. 0 (Down) ,1(Up))

Example:
state=se.GetState(id)

8.12 GetColor()

Purpose: Obtains color that corresponds to a particular status.
Synopsis:

- GetColor(status)

Arguments:

- String – status

Return value:

- String – color (e.g. red,blue,#d1f4c3 …)

Example:
color=se.GetColor("Bad")

8.13 GetType()

Purpose: Obtains type of a particular monitored object.
Synopsis:

- GetType(id)

Arguments:

- String – id of a particular monitored object

Return value:

- String – type (e.g. Hardware,Webpage,…)

Example:
type=se.GetType(id)

8.14 GetIcon()

Purpose: Obtains file name of the icon that corresponds to a particular type of monitored object.
Synopsis:

- GetIcon(type)

Arguments:

- String – type

Return value:

- String – file name

Example:
icon=se.GetIcon("FileSystem")

8.15 GetHistory()

Purpose: Obtains list of events, alarms or actions for a particular monitored object.
Synopsis:

- GetHistory(id,what)

Arguments:

- String - id of a particular monitored object
- String – what: type of message (Event, Alarm or Action)

Return value:

- List of Strings – list of messages

Example:
alist=se.GetHistory("Alarm",id)

8.16 GetPendingAction():

Purpose: Obtains list of pending actions
Synopsis:

- GetPendingAction()

Arguments:

- None

Return value:

- List of Strings – list of pending actions

Example:
palist=se.GetPendingAction()

8.17 HandlePendingAction()

Purpose: Initiates a request to perform or cancel pending action a list of pending actions.
Synopsis:

- HandelPendingAction(what,list)

Arguments:

- String – what : type of action (Perform, Cancel)
- List of Strings – list of chosen pending action

Return value:

- None

Example:

se.HandlePendingAction(“Cancel”,[“…”,”….”])

8.18 AckHistory()

Purpose: Acknowldege of chosen messages.

Synopsis:

- AckHistory(what,list)

Arguments:

- String – what : type of messages (Event,Action,Alarm)
- List of Strings – list of chosen messages

Return value:

- None

Example:

se.AckHistory(“Event”,[“…”,”….”])

8.18 Python Example of Status Engine Client

The following python module will establish communication with Status Engine with role “enstore-admin” and obtains information about root of monitored objects hierarchy, and status, color, severity level of its children . It also will get events for each child.:

```
if __name__=="__main__":
    se=SEClient("enstore-admin","user",3111,"ngop")
    root=se.TreeGetRoot()
    print "root id is ",root
    kidsList=se.TreeGetKids(root)
    for kid in kidsList:
        status=se.GetStatus(kid)
        type=se.GetType(kid)
        icon=se.GetIcon(type)
        color=se.GetColor(status)
        print "  Child %s Type %s Status %s Icon %s Color %s" %
(kid,type,status,icon,color)
        evList=se.GetHistory(kid,"Event")
        print evList
```

Chapter 9: Apache/FCGI

For efficiency, the WEB GUI for NGOP currently uses the FastCGI package to maintain a single long-running process to provide the web interface to NGOP, which avoids repeated re-connects to the Status Engine, and allows caching some information from the Status Engine.
You can run the script as a standalone CGI script, but it is noticeably slower. More details on FastCGI in general and performance are available at http://www.fastcgi.com. Other similar packages like PCGI (PersistentCGI from the Zope distribution) could be used, but some slight modification to the web_gui.py code would be needed to provide the right request-handling loop.

The Apache mod_fastcgi module is included in the Fermilab ups/upd distribution of Apache, so if you're using that distribution, no recompilation needs to be done; otherwise you can download and install mod_fastcgi (possibly as a dynamic loaded module) into your Apache configuration. Instructions on doing this are avaliable at http://www.fastcgi.com/.

We recommend using the following directives related to FastCGI in your Apache httpd.conf:
FastCgiIpcDir /var/adm/www/hostname # or wherever you keep your logs
FastCgiConfig -idle-timeout 300 -maxClassProcesses 1
And in either the httpd.conf or the .htaccess file where you place the web_gui scripts: AddHandler fastcgi-script .fcgi.
And of course, if you built FastCGI as a dynamic module, you need to precede any of the above with an appropriate LoadModule line in httpd.conf.
Properly configured, FastCGI makes the web interface much more efficient, and nicer for end users.

Chapter 10: Web Based Monitor

As of release 2.0, NGOP provides a web based monitor. The web based monitor gets information from the NGOP Central Server and provides monitoring information to the browser based on a selected *role*. A role is simply a set of system views, systems, monitoring elements and status rules that are relevant to a particular set of users.

10.1 Signing On

To access the web based gui, refer your web browser to the following URL

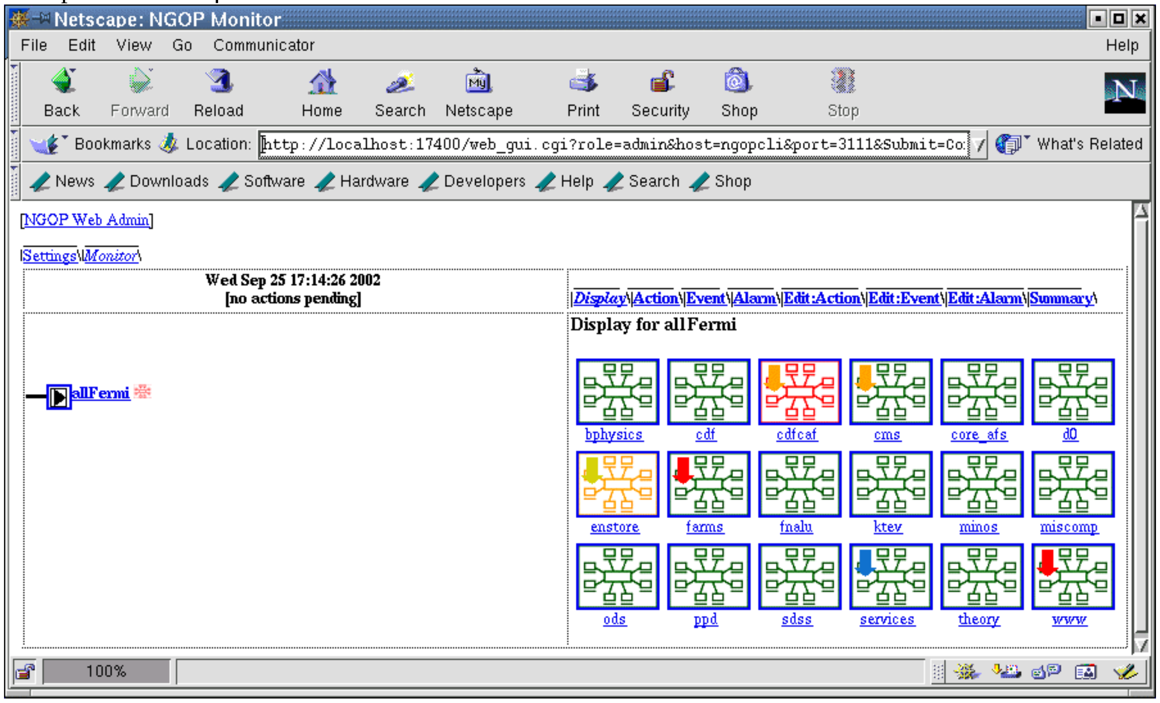
```
        https://ngopcli.fnal.gov/cgi-bin/web_gui/web_gui.fcgi
```

(Note that it is `https:` instead of the customary `http:`)

When you enter this URL, your browser will popup the login screen. You must talk to your local NGOP administrator to get a web id and password. Once you have logged in, you will be presented with a page to choose your role. Each role will have a set of objects defined that are relevant to a particular set of people.

10.2 Monitor

After the role has been chosen, the high level of monitored hierarchy will be displayed. Below is an example from the `operator` role.



http://localhost:17400/web_gui.cgi?role=admin&host=ngopcli&port=3111&Submit=Co

[NGOP Web Admin]

|Settings\|Monitor\

Wed Sep 25 17:14:26 2002
[no actions pending]

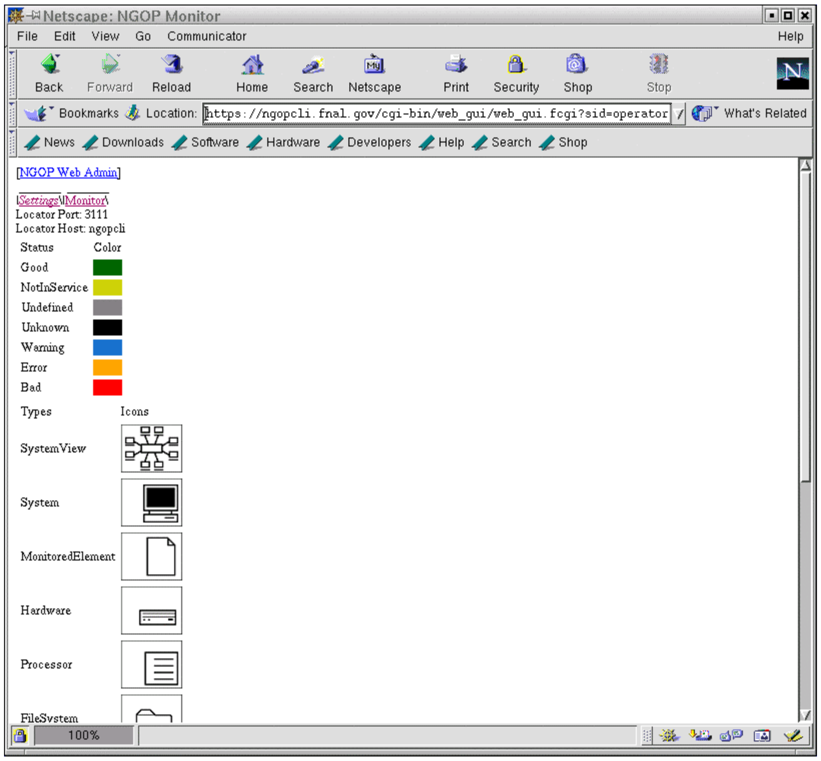
|Display\|Action\|Event\|Alarm\|Edit:Action\|Edit:Event\|Edit:Alarm\|Summary\

Display for allFermi

allFermi

bphysics	cdf	cdfcaf	cms	core_afs	d0
enstore	farms	fnalu	ktev	minos	miscomp
ods	ppd	sdss	services	theory	www

[NGOP Web Admin] – this is the link to the web admin tool that allows to modify “known” status of any of the monitored objects.
Settings – shows the current host, port information of the Locator Server as well as existing statuses, types and icons:


Netscape: NGOP Monitor
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: https://ngopcli.fnal.gov/cgi-bin/web_gui/web_gui.fcgi?sid=operator
What's Related
News Downloads Software Hardware Developers Help Search Shop
100%

[NGOP Web Admin]

\Settings\ \Monitor\
Locator Port: 3111
Locator Host: ngopcli

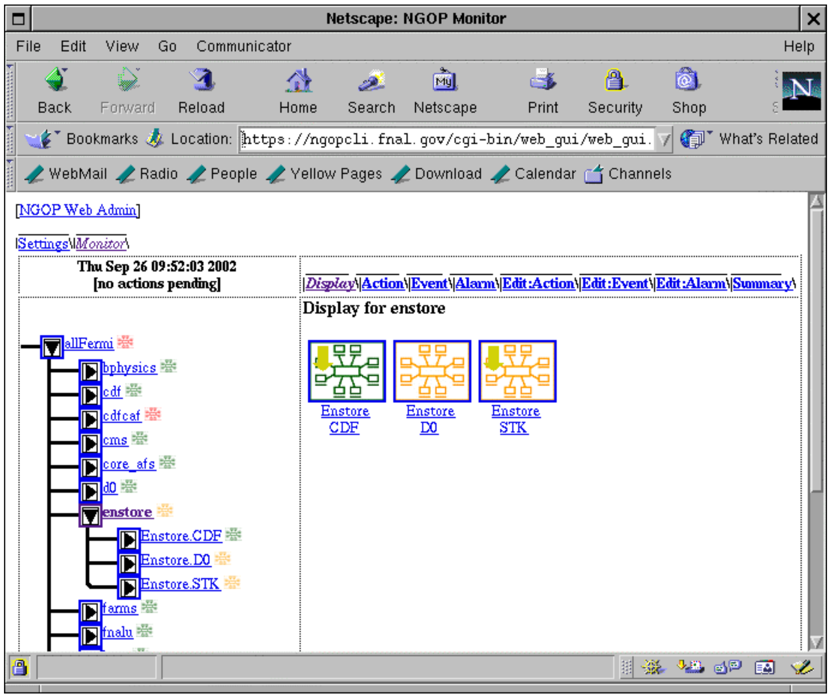
Status	Color
Good	🟩
NotInService	🟨
Undefined	▩
Unknown	⬛
Warning	🟦
Error	🟧
Bad	🟥

Types	Icons
SystemView	
System	
MonitoredElement	
Hardware	
Processor	
FileSystem	

Monitor - There are four sections to this page. The upper left corner displays the date and time of the last update from the NCS, and also states if there are any pending actions.

In the lower left corner is a concise, text based view of the display of the system. The highest level element, `allFermi` in this example, is displayed with a small red icon that represents system view. The colored icon represents the status of the monited object, which is determined by the corresponding status rules. In this case, the system view `cms` has a sever problem (red), and the enstore system view has a serious problem (yellow). Clicking the arrow key to the left of the `allFermi` text will expand the view. Note that when expanding or contracting a view, the display on the right is also affected. Likewise, drilling down on an icon in the lower right portion will affect the condensed view on the lower left.

The lower right portion contains the display for the `allFermi` system view. Each monitored object defined as part of the `allFermi` system view is displayed as a colored icon representing the state of that system, and possibly a colored arrow indicating an alarm and it's severity level on that object. It is possible to drill down into a system by clicking on it's icon. For example, clicking on the `enstore` icon would display the next level of hierarchy and provide more information.


Netscape: NGOP Monitor
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop
Bookmarks Location: https://ngopcli.fnal.gov/cgi-bin/web_gui/web_gui.
What's Related
WebMail Radio People Yellow Pages Download Calendar Channels
allFermi
bphysics
cdf
cdfcaf
cms
core_afs
d0
enstore
Enstore.CDF
Enstore.D0
Enstore.STK
farms
fnalu
Enstore CDF
Enstore D0
Enstore STK

[NGOP Web Admin]

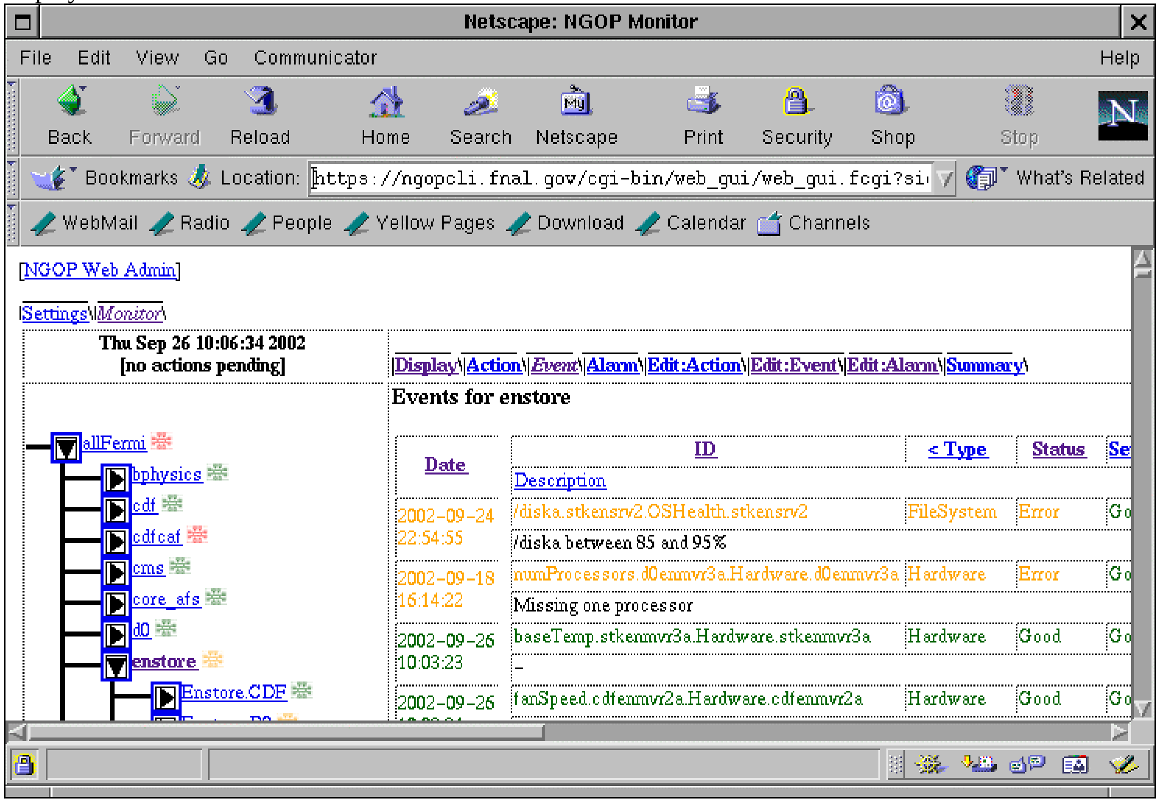
|Settings\|*Monitor*\

Thu Sep 26 09:52:03 2002
[no actions pending]

|*Display*\|Action\|Event\|Alarm\|Edit:Action\|Edit:Event\|Edit:Alarm\|Summary\

Display for enstore

The upper right portion of the window contains a menu with the items `Display, Action, Event, Alarm, Edit:Action, Edit:Event, Edit:Alarm, and Summary.` This menu controls what is displayed underneath. For example, if we selecrt Event menu item the following page will be displayed:


Netscape: NGOP Monitor
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: https://ngopcli.fnal.gov/cgi-bin/web_gui/web_gui.fcgi?si
What's Related
WebMail Radio People Yellow Pages Download Calendar Channels
allFermi
bphysics
cdf
cdfcaf
cms
core_afs
d0
enstore
Enstore.CDF

[NGOP Web Admin]

|Settings\|*Monitor*\

Thu Sep 26 10:06:34 2002
[no actions pending]

|Display\|Action\|*Event*\|Alarm\|Edit:Action\|Edit:Event\|Edit:Alarm\|Summary\

Events for enstore

Date	ID	< Type	Status	Se
	Description			
2002-09-24 22:54:55	/diska.stkensrv2.OSHealth.stkensrv2	FileSystem	Error	Go
	/diska between 85 and 95%			
2002-09-18 16:14:22	numProcessors.d0enmvr3a.Hardware.d0enmvr3a	Hardware	Error	Go
	Missing one processor			
2002-09-26 10:03:23	baseTemp.stkenmvr3a.Hardware.stkenmvr3a	Hardware	Good	Go
	-			
2002-09-26	fanSpeed.cdfenmvr2a.Hardware.cdfenmvr2a	Hardware	Good	Go

This screen simply displays the list of events for all monitored objects that belong to enstore system view. To acknowledge an event, select the `Event:Edit` link in the upper right hand corner. This will bring up a screen with the events for that monitored object, along with an Ack button for each event. After the appropriate acknowledgement buttons have been selected, click the `Apply` button to actually acknowledge the selected events. The `Check All` button will cause all alarms to be `Ack`'ed. Alarms and Actions have a similiar mechanism for display and acknowledgement. One can sort a column by clicking on the column header.
One can request to get Update information (the current value of a particular object) for system or monitored element; in order to do so click on icon located near system or monitored element name.

10.2 Standalone Web Monitor Starting/Stopping

You can start standalone web monitor….

In order to do so you have to issue the following command:
ngop web_monitor –c cfg.xml &
Configuration template file is shown below:

```
<?xml version='1.0'?>
<!DOCTYPE webmonitor_cfg SYSTEM “monitor.dtd”>
<webmonitor_cfg>
   <WebGui Type="-standalone"/>
   <LS Port="3111" Host="ngop”/>
</webmonitor_cfg>
```

Where WebGui tag required one attribute Type that can assume the following values: “-fcgi”,”-cgi”,”-standalone” . LS tag defines host and port of Location server. . This configuration file should conform to the DTD rules.

Chapter 11: Java Based Monitor

The Java GUI is a new component written as a replacement for the older python GUI. It provides graphical hierarchical representation of the NGOP monitored elements. It is written entirely in java (requires java run time environment version 1.4+) including its communication layer. It obtains all it status information from the Status Engine (and Locator Server).

11.1 Java Monitor Starting/Stopping

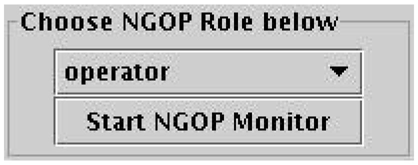
In order to do so you have to issue the following command:
ngop jmonitor –c cfg.xml &
Configuration template file is shown below:

```
<?xml version='1.0' standalone="yes"?>
<!-- DOCTYPE jmonitor_cfg SYSTEM "jmonitor.dtd" -->
<!DOCTYPE jmonitor_cfg >
<jmonitor_cfg>
   <ls Port="3111" Host="ngopcli.fnal.gov"/>
</jmonitor_cfg>
```

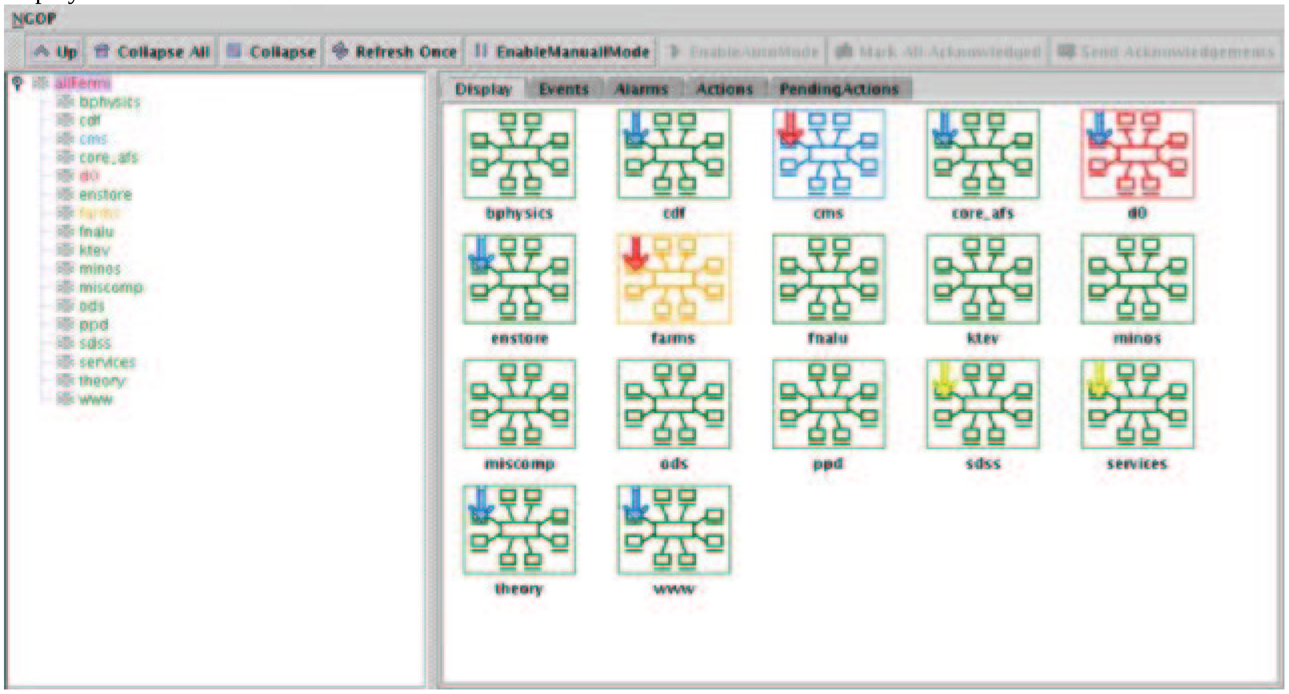
Where LS tag defines host and port of Locator Server. . This configuration file should conform to the DTD rules.

11.2 Monitor Overview

On startup one will see the role selection dialog:



After choosing the role one can click on “Start NGOP Monitor” button and the following window will be displayed:



One can browse the tree by clicking on the tree nodes or elements of the Display tab. The (default) Display tab will show the selected elements. Right clicking on the Display tab elements will show a detach/update menu for the elements that allow the operations. "Up", "Collapse All" and "Collapse" buttons affect the way the tree is displayed. It may take 10-15sec to refresh the views depending on the machine and system load or the amount of information to be retrieved.
The other tabs: Events, Alarms, Actions (and currently not implemented Pending Actions) show tables of text reverse sorted by time. The sort order can be (temporarily for now) changed by clicking or "double clicking" (or "shift clicking") on the table headers. The refreshing (which will restore the time sort) can be suspended by pressing the "EnableManuallMode" button. "EnableAutoMode" button will restore the automatic refreshing of the information.
One can acknowledge events/actions by using individual check boxes and "Mark All Acknowledged" and "Send Acknowledgements" buttons.
One exits the jmonitor by either closing the window or using the NGOP->Exit menu (or by Control-c or equivalent which is a non-confirm exit path)

Chapter 12: Configuration File Manager

The configuration files for the NGOP system are monitored by a separate set of processes referred to as the Configuration File Manager System (CFMS) which cooperate:

- Configuration file Librarian
- Configuration file Indexer
- Configuration file Broker
- Administrative client
- Monitoring Client

Each of these packages has distinct responsibilities and is described below.

12.1 Librarian

The librarian is responsible for maintaining the master copies of the configuration files. In addition, the librarian is responsible for:

- Authenticating that users have permissions for file modification
- Maintaining revision history of files to allow checkpointing, rollback, commit, and full revision history.
- Delivering the contents of particular configuration files to Monitoring Clients, the NCS, and the CFM Indexer.

The librarian uses CVS to store configuration files.

12.2 Indexer

The Indexer reads a CVS tagged set of configuration files from the Librarian, generates an index listing of the files needed for each component of the system. The indexer also performs syntax and basic sanity checks of the configuration files, as well as finding dependencies.

12.3 Broker

The Broker communicates with two types of clients, and has distinct responsibilities for each:

Monitoring /Action Servers

> When the monitoring client connects to the Broker, it sends the broker a subscription list of components. The Broker uses the indices generated by the Indexer to repeatedly send a revision tag and list of configuration files to the Monitoring Client—once initially, and then again as new indices are created. The monitoring client then requests those configuration files directly from the Librarian.

Administrative Clients

> The Broker accepts requests from the Administrative Client including a version control/rollback tag after the admin client has run the indexer and checked the new index is with the Librarian.

12.4 CFMS Configuration File

CFMS configuration file contains the following information:

```
<client_cfg>
        <Client port=”8080” host=”ngop” name=”CnfgClnt”/>
        <CfgXml cvsRep=”configxml”
                cvsRoot=”:pserver:ngop@ngop.fnal.gov:/home/ngop/Repository”
                version=”v2_0” name=”CfgXmlAC”/>
</client_cfg>
```

The <client_cfg> tag client tag defines parameters that are required to start CFMS. This tag is required and it includes the following attributes: TCP Port to connect to CFMS, host name where CFMS is running, and the name of CFMS (CnfgClnt).

The `CfgXml` tag is required. It defines the parameters that will be used to create a local configuration and connect to the CVS repository. The `CfgXml` tag includes the following attributes: name of ngop configuration cvs repository, CVSROOT definition, tag of current configuration version and the name. This configuration file should conform to the DTD rules.

12.5 CFMS Starting/Stopping

The CFMS is started in several ways. If the CFMS configuration file is located in /var/ngop/cfms directory, it is started issuing the following command:

```
    ngop start cfms
```

To start the CFMS with your own configuration file, use the following command:

```
    ngop cfms –c config_file
```

12.6 Administrative Client

The administrative client allows one to:

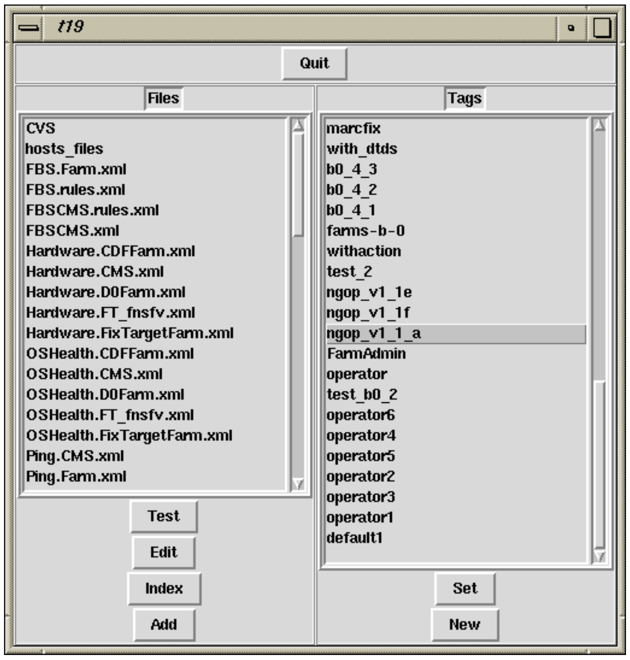
- modify/create one or more configuration files (via the Librarian, and an appropriate editor)
- Run consistency checks on the files (by invoking Indexer)
- Commit a set of changed files (possibly yielding a CVS tag)
- Notify the Broker that the new tagged version of configuration files are available.

12.7 Admin Starting/Stopping

The Admin GUI is started issuing the following command:

```
ngop admin –c config_file
```

The Admin GUI uses the same configuration file as CFMS. Below is a screen sample from the Admin GUI.



CVS
hosts_files
FBS.Farm.xml
FBS.rules.xml
FBSCMS.rules.xml
FBSCMS.xml
Hardware.CDFFarm.xml
Hardware.CMS.xml
Hardware.D0Farm.xml
Hardware.FT_fnsfv.xml
Hardware.FixTargetFarm.xml
OSHealth.CDFFarm.xml
OSHealth.CMS.xml
OSHealth.D0Farm.xml
OSHealth.FT_fnsfv.xml
OSHealth.FixTargetFarm.xml
Ping.CMS.xml
Ping.Farm.xml

marcfix
with_dtds
b0_4_3
b0_4_2
b0_4_1
farms-b-0
withaction
test_2
ngop_v1_1e
ngop_v1_1f
ngop_v1_1_a
FarmAdmin
operator
test_b0_2
operator6
operator4
operator5
operator2
operator3
operator1
default1

Chapter 13: Archive Server

13.1 Archive Server Overview

The Archive/History Server System is responsible for storing and retrieving messages generated by the NGOP system. Each message sent to the archive server is stored in an Oracle database. There are four major components of the Archive Server:

- Server: This process runs on an Oracle client machine and accepts messages from the NCS. It immediately caches the requests to local disk.
- Database Interface: This process takes the requests that have been cached by the server and stores them in the database. Having a separate process to store the data in the database allows the server to continue to run even if there are problems with the Oracle database.
- Web Interface: The information in the database is retrieved using a web-based interface.
- Cleanup Process: This process processes records in the Oracle database and rolls messages off that are more than 15 days old.

13.2 Archive Server Starting/Stopping

The script to start the Archiver is located in the `$NGOP_DIR/prototype/archsrv/src/server` directory. $NGOP_DIR is set with UPS by issuing `setup ngop`**.** To start the archiver daemons:

```
        setup ngop
        cd $NGOP_DIR/prototype/archsrv/src/server
        start_daemons
```

The `start_daemons` script launches two other scripts: `start_archiver` and `start_dbinter`**.** These scripts sit in a loop and periodically check to make sure the daemons are running. If for some reason the daemons die, the scripts will restart them.

13.3: Archiver Configuration

Below is an example configuration file for the archive server:

```
<ArchiverConfig>
<Port>7001</Port>
<ArchiverHost>fncduh1.fnal.gov</ArchiverHost>
        <OraUser>oracle_user</OraUser>
        <OraPW>oracle_pw</OraPW>
        <OraInstance>procdev</OraInstance>
<LogPath>/home/fncduh/ngop/serverlog/log.out</LogPath>
<DBInterSleepInterval>15</DBInterSleepInterval>
<RequestDirectory>/home/fncduh/ngop/scratch</RequestDirectory>
<ErrorDirectory>/home/fncduh/ngop/errors</ErrorDirectory>
</ArchiverConfig>
```

The `Port` and `ArchiverHost` are the port and host that the archive server is listening on for requests. The `OraUser/OraPW` is the Oracle userid and password of an owner that can write into the archive tables. `OraInstance` is the Oracle instance that the tables reside in. `LogPath` points to the file that contains the log files created by the archiver. `DBInterSleepInterval` is the time interval in seconds that the database interface program will look in `RequestDirectory` to process new messages. Messages are placed in `RequestDirectory` as they are received by the archive server. `ErrorDirectory` is the pathname where requests that could not be processed are placed. This configuration file should conform to the DTD rules.

Chapter 14: Monitoring Agents

14.1: Overview

Monitoring Agents (MA) are processes that monitor some entity and report a status to the NCS. NGOP provides a basic set of MA's, but users are free to write their own. The MA is the element that gives NGOP a great deal of flexibility.

A Monitoring Agent(MA) includes the following features:

- interfaces to NCS
- monitors the characteristics of a particular monitored object
- sends events to the NCS when characteristic of the object meets specific conditions (An MA doesn't send an event when the monitored object doesn't meet any conditions. In this case the State of the monitored object is assumed to be UP. A MA will send an event if a monitored object satisfies some condition.
- performs local actions
- sends requests to perform centralized actions
- sends heartbeats to the NCS
- resends events and configuration when the connection with the NCS is interrupted
- MA configuration, conditions and actions associated with conditions are described in the MA configuration file using XML. This file should be located on the node where the MA is running.

NGOP provides a framework for creation of the MAs**:** either by using the MA API or the PlugIns Agent.

14.2 Plugin Agent

A PlugIn Agent provides the monitoring of software or hardware components utilizing existing scripts or executables (plug-ins). These plug-ins should be able to measure and print some quantitative characteristics of the monitored objects. A configuration file describing the monitored hierarchy, plug-ins and a set of conditions is required in order to use a PlugIns Agent. This configuration file should conform to specific DTD rules. A configuration file should start with the following XML declarations:

```
<?xml version=”1.0”?>
<!DOCTYPE MA-config SYSTEM “agent.dtd”>
```

The first tag of a PlugIns Agent XML document is a <`MA_config`> tag, which defines the MA configuration. This tag is required, and requires additional attributes:

- `Name` – the name of the PlugIn Agents.
- `Update` – specifies the time interval in seconds between running the plug-in agent.
- Type – specifies the type of the MA. There are two possible types:
    - Daemon (default) - Monitoring Agent that should be always present
    - Cron – Monitoring Agent that will run for a short period of time and then reappear within the time interval specified in the heartbeat attribute

An <`NCS`> tag (required) defines the NCS parameters and includes the following required attributes:

- `Port` - the NCS UDP port
- `Host` - the NCS host
- `Heartbeat` - specifies the heartbeat interval in seconds

A system description should follow the <NCS> tag. Several systems are described in the same XML document. A <System> tag indicates the beginning of the system definition. It contains multiple monitored elements.

A <ConditionSet> tag indicates the beginning of the condition set definition and may be placed within a <MonitoredElement> or <System> tag. The <ConditionSet> tag contains the description of a plug-in and at least one condition.

The <fn> tag describes a plug-in function that will be executed to define the state of a monitored element. The <fn> tag has the following required attributes:

- Name - the name of the operation ("plug_in" for all the PlugIns Agents)
- Arg – the full path to the plug-in that needs to be executed to verify the state of monitored object (Parameters %ID, %Name, %Cluster and %Host are used in an attribute Arg and will be substituted with the corresponding values of monitored object)
- RetVal – the description of return values. It has the following format:
  "type:var_name, type:var_name…"
  where type is float**,** int**,** string, array int, array float or array string

Important: in case of int, float or string types the return values should be returned in the standard output of a plug-in, and separated by a newline character. In case of an "array …" type the return values should be returned in the standard output of a plug-in, and separated by space (see Example for more details). If the plug-in exits with non-zero exit code then the return value is set to Error and the following event will be generated:

```
"Date=… ID=… EventType=" executable" EventName="plug-ins" State=-2 Description="Failed to
execute command"
```

The <Condition> tag indicates the beginning of the condition definition and has the following attributes:

- State (required) - defines the monitored object state if the occurred event satisfied this condition
- SevLevel (required) - severity level of the event that satisfied this condition
- Description (required) - readable description of the event(Parameters %ID, %Name, %Cluster ,%Host and %Event is used in an attribute *Description* and will be substituted with the corresponding values of monitored object)
- EventName *(*optional) - defines the event name if it is different from the monitored object name (see Event)
- EventType (optional) - defines the event type if it is different from the monitored object type (see Event)

The <apply> tag indicates the beginning of a mathematical expression that should be evaluated in order to determine if the condition is satisfied. If an expression is evaluated to be true a PlugIn Agent will generate an event. A special variable %len(retValue_array_name) is used in a <ci> tag. It refers to the length of the array in plug-ins return value and is used in <sum>, <min> and <max> function operators (see Example for more details).

The <Action> tag is optional and it indicates the beginning of action definition. If the condition is satisfied and the action is defined, then the PlugIns Agent will perform this action locally or send a request to the NCS to execute this action.

The general structure of a PlugIns Agent configuration file should look like the following:

```
<?xml version="1.0" ?>
<!DOCTYPE MA-Config SYSTEM "agent.dtd" >
```

```
<MA-Config Name….>
 <NCS  Host=…/>
  <System Name=..>
    <MonitoredElement Name=…>
     <ConditionSet>
        <fn Name=”plug-ins” Arg=…/>
        <Condition State=…>
          <apply>
                <!—expression--!>
          </apply>
          <Action>
                <!—action--!>
          </Action>
        </Condition>
        <!—more conditions--!>
     </ConditionSet>
    </MonitoredElement>
    <!—more monitored elements--!>
  </System>
  <!—more systems--!>
</MA-Config>
```

In the PlugIns Agent configuration `localhost` instances will be replaced by the local host name. This configuration file should conform to the DTD rules.

Example 1:
Let’s assume that we want to monitor the system load averages for the past 1, 5, and 15 minutes using the following command as a “plugin”:

```
uptime|awk ‘{print $(NF-2),$(NF-1),$NF}’|awk -F’,’ ‘{print $1,$2,$3}’
```

Assume that we want the PlugIns agent to generate an event when the minimum of cpu load averages exceeds 12.0. The fragment of the PlugIns configuration file to perform this would look like the following:

```
<MonitoredElement Name=”cpuLoad”  Host=”localhost” Type=”sysUsage”>
<ConditionSet>
<fn Name=”plug_ins” Arg=” uptime|awk ‘{print $(NF-2),
               $(NF-1),$NF}’|awk -F’,’ ‘{print $1,$2,$3}’ “
               RetVal=”array float:load”/>
   <Condition State=”UP” SevLevel=”6” Description=”Cpu load too high”>
   <apply>
        <geq/>
        <apply>
          <min>
               <bvar>i<bvar>
               <lowlimit><cn>0<cn></lowlimit>
               <uperlimit><ci>%len(load)</ci></uperlimit>
               <ci>load</ci>
          </min>
        </apply>
        <cn>12.0</cn>
  </apply>
  <!—checking for condition : min(load[i]) >=12.0, where i = 0, len(load)--!>
 </Condition>
</ConditionSet>
```

Example 2:

Let’s assume that we want to monitor OS “Health” on an SGI node named fnsfo. We want to check some components using the following Unix commands:

Number of cpu off –line:

```
mpadmin -n|wc -l
```

Cpu load during last 15 min:

```
        uptime | awk -F’,’ ‘{print $NF}’
```
/dev/root file system size
```
        df /dev/root | grep -v Filesystem|awk ‘{print $6}’
```
Inetd daemon presence
```
        ps -ef | grep inetd | grep -v grep|wc –l
```

The agent’s configuration file will look like the following:

```
<?xml version=”1.0”?>
<!DOCTYPE MA-conifg System “agent.dtd”>
<MA-config Name=”SGI_Health” Update=”180”>
<NCS Heartbeat=”600” Port=”19997” Host=”ndem.fnal.gov”/>
    <System Name=”OSHealth” Cluster=”localhost”>
    <!—system id is “OSHealth.fnsfo” --!>
       <MonitoredElement Name=”cpuStatus”  Host=”localhost” Type=”Hardware”>
       <!—monitored element id  is “cpuStatus.fnsfo.OSHealth.fnsfo” --!>
<ConditionSet>
<fn Name=”plug_ins” Arg=”mpadmin -n|wc -l” RetVal=”int:onlinecount”/>
  <Condition State=”Down” SevLevel=”6” Description=”At least one cpu is off-line”>
       <apply>
               <eq/>
               <ci>onlinecount</ci>
               <cn>4</cn>
       </apply>
       <!—checking for condition : (onlinecount == 4) , where number of
                      processors on fnsfo is equal to 4--!>
   </Condition>
</ConditionSet>
</MonitoredElement>
<MonitoredElement Name=”cpuLoad”  Host=”localhost” Type=”sysUsage”>
<!—monitored element id  is “cpuLoad.fnsfo.OSHealth.fnsfo” --!>
<ConditionSet>
<fn Name=”plug_ins” Arg=”uptime | awk -F’,’ ‘{print $NF}’”  RetVal=”float:load”/>
  <Condition State=”UP” SevLevel=”4” Description=”Cpu load is  between 8  and 15 during
                                                                   last 15 minutes”>
       <apply>
       <and/>
         <apply>
                 <geq/>
                 <ci>load</ci>
                 <cn>8.0</cn>
        </apply>
        <apply>
                 <lt/>
                 <ci>load</ci>
                 <cn>15.0</cn>
         </apply>
       </apply>
       <!—checking for condition : ((load>=8)&& (load<15)) --!>
  </Condition>
  <Condition State=”UP” SevLevel=”6” Description=”Cpu load is greater than 15  during last
15 minutes”>
       <apply>
                 <geq/>
                 <ci>load</ci>
                 <cn>15.0</cn>
       </apply>
       <!—checking for condition : (load>=15) --!>
  </Condition>
</ConditionSet>
</MonitoredElement>
<MonitoredElement Name=”/dev/root”  Host=”localhost” Type=”FileSystem”>
<!—monitored element id  is “/dev/root.fnsfo.OSHealth.fnsfo” --!>
<ConditionSet>
<fn Name=”plug_ins” Arg=”df /dev/root | grep -v Filesystem|awk ‘{print $6}’”
       RetVal=”int:size”/>
  <Condition State=”UP” SevLevel=”6” Description=”file system is more then 95% full”>
   <apply>
               <gt/>
               <ci>size</ci>
               <cn>95</cn>
   </apply>
   <!—checking for condition : (size>95%) --!>
   </Condition>
</ConditionSet>
</MonitoredElement>
<MonitoredElement Name=”inetd”  Host=”localhost” Type=”Daemon”>
<!—monitored element id  is “inetd.fnsfo.OSHealth.fnsfo” --!>
<ConditionSet>
  <fn Name=”plug_ins” Arg=”ps -ef | grep inetd | grep -v grep|wc -l”
                                                           RetVal=”int:ifExist”/>
     <Condition State=”Down” SevLevel=”6” Description=”inetd daemons is not running”>
       <apply>
              <neq/>
              <ci>ifExists</ci>
              <cn>1</cn>
       </apply>
     </Condition>
</ConditionSet>
</MonitoredElement>
</System>
</MA-config>
```

14.2.1 Starting/Stopping Plugins Agent

You can start a Plugins Agent in several ways. All Plugins Agent configuration files are placed in /var/ngop/plugins_agent directory and may be started/stopped simultaneously by issuing

```
    ngop start/stop  plugins_agent
```

In order to start individual agents, the following commands are used:

```
        ngop start/stop  “ngop plugins_agent –c /ngop/var/plugins_agent/cfg_file”
```

or

```
        ngop plugins_agent –c cfg_file
```

(you have to manually kill an agent if started it this way)

14.3 Ping Agent

NGOP comes packaged with a Ping type monitoring agent. The Ping Agent periodically sends ICMP packets to nodes listed in it’s configuration file. It is also can perform route discovery and has an ability to distinguish failure to ping the node from the failure to ping the switch, as well as discovery of simultaneous multiple failures. In addition, if the remote machine is running the `rstatd` daemon, the Ping Agent can determine the boot time of a node as well as it’s cpu load.

A template (`ping.xml`) configuration is supplied with NGOP.   The primary function call that is made to determine if a node is up is appropriately called `isUp`. This function takes two optional arguments (time delayed in minutes before the node will be decalred as “unpingbale”; rsatd flag that inidicates either the attempt to connect to remote rstatd daemon should be made). “isUp” function returns an integer value. The values returned reflect the various conditions that could be encountered when setting up and sending an ICMP request to a node. These values are as follows:

- 0: the machine is Up
- 1: An ICMP request failed twice in the past N minutes, where N is defined as the update interval.
- 2: The ICMP request resulted in lost packets
- 3: The request to create an ICMP socket failed.
- 4: The machine that is being pinged has been rebooted. This information can explain why a machine was returning condition 1.
- 5: The ICMP request timed out
- 7: Unknown machine
- 8: The machine is unreachable as well as at least N other machines in the same cluster *

- 9: The machine is unreachable because of the network problems *

* - is value could be returned only if isNetworkDown is used
To obtain remote performance values from `rstatd`, the function `getLoad` is provided. This function returns a floating point number that represents the CPU load on that machine.

To perform route discovery and determine multiple failueres the “isNetworkDown” function is provided. This function takes two arguments (the time interval betwen recteating the route table in min, and the threshold that defines teh notion of “multiple” failures). This function is applied to an entire cluster rather than to a particular host.

The table below is based on the pre-packaged configuration file for the Ping Agent.

Function Name	**Condition**	**Event Value**	**State**	**Sev Level**	**Description**	**Action**
IsNetworkDown	Multiple nodes (> N) became unpingbale during last M min, but network has no problems	nodes:node1,node2,... nodeN	1	6	Multiple nodes are unpingable. Type:nodes: node1, node2 ...	email
	Multiple nodes (> N) became unpingbale during last M min, but it happened because of some switches	switches: switch1,switch2	1	6	Multiple nodes are unpingable. Possilble network problems! Type:switches: switch1, switch2 ...	email
isUp	Ping failed during last N minutes	1	0	2	Host is unpingbale	email
	Lost Packets	2	1	4	The ICMP request resulted in lost packets	
	Failed to create raw socket	3	1	4	ICMP service is not available	
	Stored boot time differs than actual	4	1	5	Host was rebooted	email
	Ping failed	5	1	4	Ping timed out	
	Machine name is unknown	7	1	0	Machine name is unknown	
	Ping failed, but multiple nodes are unpigbale as well	8	0	2	Hos is unpingbale	
	Ping failed because of network	9	0	2	Host is unpingable	

	problems					
getLoad	CPU Load	20.0	1	6	CPU load high	email

The following is the XML code that will implement the above table:

```
<MA-config Update="180" Name="PingAgentMyCluster" Type="Daemon">
   <NCS Heartbeat="300" Port="19997" Host="localhost" />
   <System Name="Ping" Cluster="MyCluster">
         <ConditionSet>
            <fn Name="isNetworkDown" Arg="Refresh=720,Counter=2"
RetVal="str:Type,str:Names"/>
            <Condition State="1"  SevLevel="5" Description="Multiple
nodes are unreachable!Possible network problem!" EventName="switch"
EventType="Network">
               <apply>
                  <apply>
                     <eq/>
                     <ci>Type</ci>
                     <cn>"switches"</cn>
                  </apply>
               </apply>
                <Action ID="email" Host="localhost" Type="central">
            <Exec Name="email"
Argument="Address:address,Subject:Ngop_Report,Message:%Cluster:%ServiceT
ype:%ID:%Description%EventValue are unreachable" />
        </Action>
          </Condition>
        <Condition State="1"  SevLevel="6" Description="Multiple nodes
are unreachable!" EventName="nodes">
               <apply>
                  <apply>
                     <eq/>
                     <ci>Type</ci>
                     <cn>"nodes"</cn>
                  </apply>
               </apply>
                <Action ID="email" Host="localhost" Type="central" >
           <Exec Name="email"
Argument="Address:address,Subject:Ngop_Report,Message:%Cluster:%ServiceT
ype:%ID:%Description%EventValue are unreachable"  />
        </Action>
          </Condition>
      </ConditionSet>
      <For Each="Host" In="Cluster" Name="ClusterA" Var="{%Host}"
Filename="hostsInClusters.xml" >

     <MonitoredElement Name="ping" Type="Hardware" Host="{%Host}">
         <ConditionSet>
            <fn Name="isUp" Arg="Delay=3,Rstatd=1" RetVal="int:x"/>
            <Condition State="1"  SevLevel="5" Description="Host was
rebooted">
               <apply>
                  <apply>
                     <eq/>
                     <ci>x</ci>
```

```
                    <cn>4</cn>
                 </apply>
              </apply>
           <Action ID="email" Host="localhost" Type="central" >
           <Exec Name="email"
Argument="Address:address,Subject:Ngop_Report,Message:%Host:%ServiceType
:%ID:%Description" />
        </Action>
           </Condition>

           <Condition State="0" SevLevel="2" Description="Host is
unreachable">
              <apply>
                 <apply>
                    <eq/>
                    <ci>x</ci>
                    <cn>1</cn>
                 </apply>
              </apply>
           <Action ID="email" Host="localhost" Type="central" >
           <Exec Name="email"
Argument="Address:address,Subject:Ngop_Report,Keyword:NodeUnusable,Messa
ge:%Host:%ServiceType:%ID:%Description" />
        </Action>
           </Condition>
          <Condition State="0" SevLevel="2" Description="Host is
unreachable,possible problem with network">
              <apply>
                 <apply>
                    <eq/>
                    <ci>x</ci>
                    <cn>9</cn>
                 </apply>
              </apply>
           </Condition>
           <Condition State="0" SevLevel="2" Description="Host is
unreachable for">
              <apply>
                 <apply>
                    <eq/>
                    <ci>x</ci>
                    <cn>8</cn>
                 </apply>
              </apply>
           </Condition>
           <Condition State="1"  SevLevel="4" Description="Packets
lost">
              <apply>
                 <apply>
                    <eq/>
                    <ci>x</ci>
                    <cn>2</cn>
                 </apply>
              </apply>
           </Condition>
           <Condition State="1" SevLevel="4" Description="ICMP Service
is unavailable">
              <apply>
```

```
                  <apply>
                     <eq/>
                     <ci>x</ci>
                     <cn>3</cn>
                  </apply>
               </apply>
            </Condition>
            <Condition State="1" SevLevel="4" Description="Ping timed
out">
               <apply>
                  <apply>
                     <eq/>
                     <ci>x</ci>
                     <cn>5</cn>
                  </apply>
               </apply>
            </Condition>

         </ConditionSet>
      </MonitoredElement>
      </For>
      </System>
      <For Each="Host" In="Cluster" Name="ClusterA" Var="{%Host}"
Filename="hostsInClusters.xml" >
       <System Name="OSHealth" Cluster="{%Host}">
      <MonitoredElement Name="cpuLoad" Type="sysUsage" Host="{%Host}">
         <ConditionSet>
            <fn Name="getLoad" Arg="" RetVal="float:load"/>
                <Condition State="1" SevLevel="6" Description="Average
cpu load during last 15 min exceeds 4 on the %Host">
                <apply>
                <gt/>
                <ci>load</ci>
                <cn>4</cn>
                </apply>
            <Action ID="email" Host="localhost" Type="central" >
                  <Exec Name="email"
Argument="Address:address,Subject:Ngop_Report,Message:%Host:%ServiceType
:%ID:%Description" />
            </Action>
                </Condition>
              <Condition State="1" SevLevel="5" Description="Average cpu
load is between 4 and 8 during last 15 min">
                <apply>
                <and/>
                <apply>
                  <gt/>
                  <ci>load</ci>
                  <cn>4</cn>
                </apply>
                <apply>
                  <leq/>
                  <ci>load</ci>
                  <cn>8</cn>
                </apply>
                </apply>
                </Condition>
         </ConditionSet>
```

```
        </MonitoredElement>
        </System>
        </For>
</MA-config>
```

This configuration file should conform to the DTD rules.

14.3.1 Ping Agent Starting/Stopping

The ping agent can be started using the `ngop start` command:

```
        $setup ngop
        $ngop ping_agent –c <ping_agent xml file name>
```

In addition, if the directory `/var/ngop/ping_agent exists,` then simply issuing the following is sufficient to start the ping agent(and other agents located in `/var/ngop)`:

```
        $ setup ngop
        $ ngop start
```

14.4 Swatch Agents

A Swatch Agent is an agent that watches a log file for lines matching a regular expression, and takes some action when this occurs (similar to swatch). Like the other agents in NGOP, an XML configuration file controls the Swatch Agent's behavior. An XML configuration file for each Swatch Agent is placed in a separate file and should begin with the following XML declarations:

```
        <?xml version="1.0"?>
        <!DOCTYPE SwatchAgentConfig SYSTEM "swatchagent.dtd" >
```

The outermost tag of the file is `<SwatchAgentConfig>` , which includes the required `name` attribute. `name` specifies the name of the monitoring agent.

The second outermost tag of the file is `<NCS>`, which includes the following attributes:

- `Heartbeat` - specifies the hearbeat interval in seconds
- `Host` - specifies the host name of the NCS to send events
- `Port` - specifies the port number on the above host

The third outermost tag of the file is `<File>`, which includes the following attributes:

- `file` - This lists the file the agent should watch for messages
- `filetype` – The valid values for `filetype` are:
- `"multihost"` – This indicates that a hostname match should be prepended to regular expressions when expanding `HostType` lists.
- `"plain"` – This indicates that all regular expressions are to be used verbatim

A system description should follow the`<File>` tag. Several systems are described in the same XML document. The `<System>` tag indicates the beginning of the system definition. It contains multiple monitored elements.

Once we are in the context of a given `<MonitoredElement>,` we can specify rules about log file lines, which will trigger events about that monitored element with an **`<ReRule>`** tag. `<ReRule>` tag has the following required attributes:

- `Regexp` – defines a regular expression

```
State
SevLevel
EventName
EventValue
```

and one optional attribute:

`ActionID` – defines action that should be executed when pattern is matched
An `<Action>` tag that should be within a `<SwatchAgentConfig>` tag describes an action.

In the Swatch Agent configuration instances of "`localhost`" will be replaced by the local host name. This configuration file should conform to the DTD rules.

Example:
Let's assume that we want to monitor a `syslogd` log file on a Linux machine. We want to watch for the following patterns:

```
'kernel: nfs: server.*not responding'
'ypbind.*failed'
'shutdown succeeded'
'startup succeeded'
'kernel:.*irq
'kernel:.*reset: success'
'kernel:.*status timeout:'
'kernel:.*drive not ready for command'
```

```
<?xml version='1.0'?>
<!DOCTYPE SwatchAgentConfig SYSTEM "swatchagent.dtd" >
<SwatchAgentConfig name="SwatchAgent">
<NCS  Heartbeat="600" Host='ndem.fnal.gov' Port='19997'>
<File  file='/var/log/messages' filetype='plain'>
 <System ID='OSHealth' Cluster='localhost'>
   <MonitoredElement Name='syslogd' Type='Daemon' Host='localhost'>
     <ReRule Regexp='kernel: nfs: server.*not responding' EventName='nfs'
                    State='1' SevLevel='6'/>
     <ReRule Regexp='ypbind.*failed' EventName='ypbind' State='1' SevLevel='4'/>
     <ReRule Regexp='shutdown succeeded'  State='1' SevLevel ='5'/>
     <ReRule Regexp='startup succeeded'  State='1' SevLevel= '0'/>
     <ReRule Regexp='kernel:.*irq timeout'  State='1' SevLevel= '6'/>
     <ReRule Regexp='kernel:.*reset: success'  State='1' SevLevel= '6'/>
     <ReRule Regexp='kernel:.*status timeout:'  State='1' SevLevel= '6'/>
     <ReRule Regexp='kernel:.*drive not ready for command'  State='1'
                    SevLevel= '6'/>
     <ReRule Regexp='kernel:.*Unable to load interpreter /lib/ld-linux.so.2'
                    State='1' SevLevel= '6'/>
   </MonitoredElement>
 </System>
</File>
</SwatchAgentConfig>
```

14.4.1 Starting/Stopping Swatch Agent

You can start Swatch Agent in several ways. All Swatch Agent configuration files are placed in the /var/ngop/swatch_agent directory and could start (stop) simultaneously by issuing

```
ngop start/stop  swatch_agent
```

In order to start individual agents , the following commands are used:

```
ngop start/stop  "ngop swatch_agent –c /ngop/var/swatch_agent/cfg_file"
```

or

```
ngop swatch_agent –c cfg_file
```

The agent must be killed manually if started by the latter.

14.5 URL Agent

The URL Agent scans given URL’s for reachability. Like the other agents in NGOP, an XML configuration file controls the URL Agent’s behavior. An XML configuration file for each URL Agent is placed in a separate file and should begin with the following XML declarations:

```
<?xml version=”1.0”?>
<!DOCTYPE URLAgentConfig SYSTEM “URLagent.dtd” >
```

The outermost tag of the file is `<URLAgentConfig>` , which includes the required `name` attribute. `name` specifies the name of the monitoring agent. An optional `Scan` attribute can also be specified, which refers to the time between scans in seconds. This configuration file should conform to the DTD rules.

Example:

```
<?xml version="1.0"?>
<!DOCTYPE URLAgentConfig SYSTEM "URLagent.dtd" >

<URLAgentConfig Name="fast_URL_agent_localhost" Scan="900">

 <NCS Host="ndem.fnal.gov" Port="19997" Heartbeat="300"/>

 <!-- Items we watch every 15 minutes -->

 <Action ID="email" Local="email_miscomp" Type="central">
    <Exec Name="email" Argument="Address:miscomp@fnal.gov,
     ngop-team@fnal.gov, Subject:Ngop_Report,
     Message:%Host:%ServiceType:%ID:&quot;%Description&quot;"/>
 </Action>

 <Action ID="email" Local="email_cdweb" Type="central">
    <Exec Name="email" Argument="Address:operator@fnal.gov,
     csi-group@fnal.gov,cdweb@fnal.gov,
     ngop-team@fnal.gov, tom@pager.fnal.gov, Subject:Ngop_Report,
     Message:%Host:%ServiceType:%ID:&quot;%Description&quot;"/>
 </Action>

 <Action ID="email" Local="email_csd" Type="central">
    <Exec Name="email" Argument="Address:operator@fnal.gov,
     dick@fnal.gov,ngop-team@fnal.gov,
     Subject:NGOP-Remedy_webserver_unavailable,
     Message:Remedy_webserver_unavailable_on_%Host:%ServiceType:%ID:
     &quot;%Description&quot;"/>
 </Action>

 <Action ID="email" Local="email_ngop" Type="central">
    <Exec Name="email" Argument="Address:ngop-admin@fnal.gov,
     Subject:NGOP_URL_Report, Message:%Host:%ServiceType:%ID:&quot;%Description&quot;"/>
 </Action>

 <System Name="www" Cluster="WWW">

    <MonitoredElement Name="mainpage" Type="webpage" Host="www0">
      <URLFailRule ActionLocal="email_cdweb" Href="http://www.fnal.gov/"
        RegExp="Fermilab"
      />
    </MonitoredElement>

    <MonitoredElement Name="telephone" Type="webpage" Host="www0">
      <URLFailRule ActionLocal="email_cdweb"
         Href="http://www-tele.fnal.gov/cgi/bin/telephone.script?format=text&amp;
         Name=harry&amp; which=last&amp;exact=&amp;output=name"
        RegExp="TOMDICKANDHARRY"
      />
    </MonitoredElement>

    <MonitoredElement Name="disclaimer" Type="webpage" Host="www0">
      <URLFailRule ActionLocal="email_cdweb"
       Href="http://www.fnal.gov/pub/disclaim.html"
       RegExp="Unauthorized attempts"
      />
    </MonitoredElement>

    <MonitoredElement Name="directorate" Type="webpage" Host="www0">
      <URLFailRule ActionLocal="email_cdweb" Href="http://www.fnal.gov/directorate/"
      />
    </MonitoredElement>

    <MonitoredElement Name="faw" Type="webpage" Host="www0">
      <URLFailRule ActionLocal="email_cdweb" Href="http://www.fnal.gov/faw/"
        RegExp="Work Resources"
        />
    </MonitoredElement>

    <MonitoredElement Name="stock" Type="webpage" Host="www0">
      <URLFailRule ActionLocal="email_cdweb"

       Href="http://www-
stock.fnal.gov/cgibin/stock.script?stock_item=wrench&amp;match=and&amp;
       format=html&amp;debug=false"
        RegExp="VICE.GRIP"
        />
    </MonitoredElement>

 </System>

</URLAgentConfig>
```

14.6 Monitoring Agent API

Users can write their own monitoring agents using the supplied monitoring agent API that comes with the NGOP product. This chapter discussed the monitoring agent API and gives examples.

14.6.1: API Description

NGOP Monitoring Agent API provides way for users to write their own Monitoring Agent that will communicate with NGOP Central Server.

The MAClient Class performs all the communication between the Monitoring Agent and the NGOP Central Server. The user has to create the object of this class. The MAClient Class provides the following methods:

- Setting MA attributes
- Describing configuration
- Registering with NGOP Central Server
- Sending Events to NGOP Central server
- Performing Action
- Sending request to NCS to perform Action

Only a Python binding API is provided in the prototype version.

14.6.1.1: `MAClient` Class

In order to use the MA API, user applications should import the `MAClient` class from the `MA_API` module:

```
from MA_API import MAClient
```

14.6.1.2: `MAClient` methods

This section describes the methods available for the `MAClient` class.

15.6.1.2.1 `MAClient()`

The constructor `MAClient()` creates an `MAClient` object and establishes communication with the NGOP Central Server.

Synopsis:	`MAClient()`
Arguments:	`None`
Return Value:	`MAClient` object.

15.6.1.2.2 `setMAAttrib()`

This method sets the monitoring agent attributes such as name, heartbeat rate, central server host and port.

Synopsis:	`setMAAttrib(maName,heartbeat,ncsHost,ncsPort,type)`
Arguments:	
`maName:`	String; Monitoring Agent name
`heartbeat:`	String; interval in seconds to send a heartbeat message to the NCS.
`ncsHost:`	String; NCS host
`ncsPort:`	String; NCS port
type:	String,MA type (Cron or Daemon)
Return Value:	None.

15.6.1.2.3 `addSystem()`

This method adds system information to the list of monitored objects.

Synopsis:	`addSystem(sysName, clusterName)`
Arguments:	
`sysName:`	String; name of the system.
`clusterName:`	String; name of the cluster.
Return Value:	None.

15.6.1.2.4 `addME()`

This method adds monitored elements to the system configuration.

Synopsis:	`addME(sysName, clusterName, meName, meType, host)`
Arguments:	
`sysName:`	String; name of the system.

`clusterName:` String; name of the cluster.
`meName`: String: monitored element name.
`meType`: String: monitored element type.
`host`: String: host name where the monitored element is located.
Return Value: None.

15.6.1.2.5 `register()`

This method sends the initial configuratio to the NGOP Central Server.
Synopsis: `register()`
Arguments: None
Return Value: None

15.6.2.6 `send_event()`

This method sends an event message to the NGOP Central Server.
Synopsis: `send_event(eventDict,sysName,clusterName,meName,meHost)`
Arguments:
`eventDict:` Dictionary: Describes the event with the following dictionary keys:
`EventType` – String
`EventName` – String
`EventValue` – String
`State` – Integer (-1,0,1)
1 - undefined
0 - up
1 - down
`SevLevel` – Integer (0-6).
0 – None
1 - NotInService
2 - Unknown
3 - Undefined
4 - Warning
5 - Error
6 - Alert
`sysName:` String: name of the system
`clusterName:` String: name of the cluster
`meName:` String: name of the monitored element
`meHost:` String: name of the host where the monitored element is located.
`meName,meHost are set to None if event is related to system state`
Return Value: 2-type (`status`,`reason`)
`status`: Integer
0 – failure
1 – success
`reason`: String; Reason for failure or NULL.

15.6.1.2.6 `do_action()`

This method sends an event message to the NGOP Central Server.

Synopsis:
`do_action(sysName,clusterName,meName,meHost,eventDict,actionDict)`
Arguments:
`eventDict:` Dictionary: Describes the event with the following dictionary keys:
- `EventType` – String
- `EventName` – String
- `EventValue` – String
- `State` – Integer (-1,0,1)
  - 1 - undefined
  - 0 - up
  - 1 - down
- `SevLevel` – Integer (0-6).
  - 0 – None
  - 1 - NotInService
  - 2 - Unknown
  - 3 - Undefined
  - 4 - Warning
  - 5 - Error
  - 6 - Alert

`actionDict`: Dictionary: Describes the action with the following keys:
- `ActionID` – String: The action id.
- `ExecName` – String: Name of the command to be executed.
- `ArgList` – String: The argument list to `ExecName.`
- `ActionType` – String: Either "`local`" or "`central`".

`sysName:` String: name of the system
`clusterName:` String: name of the cluster
`meName:` String: name of the monitored element.
`meHost:` String: name of the host where the monitored element is located.
`meName,meHost are set to None if event is related to system state`

Return Value: None.

15.6.1.2.7 `stop()`

This method notifies the NCS that it ended normally .
Synopsis: `stop()`
Arguments:
None
Return Value: None

14.6.2: MA API Example

This section details a monitoring agent written using the API. In this example, let's assume that we want to monitor the system "`mySystem`" on the cluster "`myCluster`". Let's say the cluster consists of 100 nodes named `myWorker1`, `myWorker2`, ..., `myWorker100`. A monitored element called `myDaemon` is running on each the node in the cluster. When `myDaemon` dies or restarts we would like to send an event message to the NGOP Central Server.

Here is the code to perform this task:

```
import MA_API
import time
import sys
DOWN=0
UP=1
UNKNOWN=-1
def isDaemonAlive(self,name,node):
   #user provides way to verify that the daemon is alive on the node
  ......
   return state,description
   #state could be Down, Up, Unknown
   #description should not have blanks
if __name__==”__main__”:
    checkTime=myCheck
    # monitoring interval
    maName=”myAgent”
    #name of the monitoring agent
    sysName=”mySystem”
    #system name
    clusterName=”myCluster”
    #cluster name
    nodeName=”myWorker”
    #common node name
    minIdx=1
    #node number starts with this index
    maxIdx=100
    #node number ends with this index
    meName=”myDaemon”
    #name of monitored element
    meType=”Daemon”
    #type of monitored element
    heartbeat=”300”
    #heartbeat rate in sec
    serverHost=’ngop’
    serverPort=”19997”
    #NGOP Central Server host and port

    cl=MA_API.MAClient()
    #creates MAClient object

    cl.setMAAttrib(maName,heartbeat,serverHost,serverPort)
    #sets MA attributes

    cl.addSystem(sysName,clusterName)
    #configures the system
    oldStateList=[]
    #hold previous state of the monitred element
    for i in range(minIdx,maxIdx):
        cl.addME(sysName,clusterName,meName,meType,nodeName+repr(i))
        #configures system monitored elements list
        oldStateList.append(UP)
        # sets all state to UP

    cl.register()
    #registers monitoring agent with NGOP Central Server

    while 1:

        for i in range(minIdx,maxIdx):

            state,description=isDaemonAlive(meName, nodeName+repr(i))
            if oldStateList[i]==state:
              continue #nothing has changed
            eventDict={’EventType’:meType, ‘EventName’:meName, \
              ‘State’:state,’SevLevel’:0}
            eventDict[’Description’]=description

             status,reason=cl.sendEvent(eventDict,sysName,clusterName,
                                        meName,nodeName)

            #sends event to NGOP CS

             if not status: print “Error:”,reason
            oldStateList[i]=state

          time.sleep(checkTime)
```

14.6.3 Starting/Stopping Your Agent

You can start your Agent issuing the following command:

```
    ngop your_python_code.py &
```

Chapter 15: Action Server

An Action Server has the following features:

- It gets configuration information from the CFMS
- It gets action requests from the NCS
- It verifies user authorization to request the actions
- It verifies that monitored object accosiated with an anction is not marked as “known bad”
- It performs actions
- It notifies the NCS about success/failure of performed actions

There are several configuration files that contain general information needed for the Action Servers. These files will be downloaded into a designated configuration area during the NGOP Action Server startup.

15.1 Action Server Configuration File

The Action Server configuration file contains the following information:

```
<?xml version='1.0'?>
<!DOCTYPE AS_cfg SYSTEM "server.dtd">
<AS_cfg DebugLevel="3">
   <Client Port="19996" Host="ngop" Name="NCSClnt"/>
   <Client Port="8080" Host="ngop" Name="CFMSClnt"/>
   <CfgXml CvsRep="configxml" WrkDir=".ngop_action" ExcDir="scripts"
CvsRoot=":pserver:ngop@ngop.fnal.gov:/home/ngop/Repository
" Role="default"/>
   <ActionObjectList>
     <MonitoredElement Cluster="NGOP" System="NGOPService" Host="localhost"
Name="action"/>
      <Host Name=”localhost”/>
   </ActionObjectList>
</AS_cfg>
```

The AS_cfg tag has one optional attribute that defines debug level output (0 –6) of the action server log files. Two log files (ActionServer_cfgname.out and ActionServer_cfgname.err) are created automaticaly in ~/Log/ActionServer_cfgname directory, where “cfgname” is the name of configuration file. If directory doesn’t exist it will be created. Log files are rotated daily: the old files are moved to “name.timestamp” files.

Action Server should be connected to NCS, so first <`Client`> tag is required. The second <`Client`> tag is optional and is needed if Action Server connects to CFMS. The <`Client`> tag has the following attributes: service tcp port, host name of the node where service is running and service name (`NCSClnt/CFMSClnt`). The `CfgXml` tag is required. It defines the parameters that will be used to create local configuration and connect to CVS repository. `CfgXml` tag includes the following attributes: name of the root dirctory (required), name of ngop configuration cvs repository (requred), `CVSROOT` definition (optional), tag of current configuration role. The ActionObjectList tag is optional. If it is present it contains the list MonitoredElements, Hosts, Systems and Cluster tags. If at least one of the monitored objects listed here is declared as "known bad" (see known status) the all actions will be supressed until the time when the object becomes "good".

This configuration file should conform to the DTD rules.

15.2 Starting/Stopping Action Server

An administrator can start Action Server in several ways. Action Server configuration files are placed in /var/ngop/action directory and could start (stop) simultaneously by issuing

```
        ngop start/stop  action
```

or

```
         ngop action –c cfg_file &
```

If the agent is started by the latter command, the agent can only be killed manually.

15.3 File `authorized.xml`

The `authorized.xml` configuration file contains information about the users who are authorized to perform certain actions via an Action Server. Each user belongs to an authorization group.
If a user has requested an action but is not listed in the `authorized.xml` configuration file, the request will be denied. The `authorized.xml` file requires the following declaration and tags:

```
<?xml version='1.0'?>
<!DOCTYPE NGOPAction  SYSTEM "action.dtd">
<NGOPAction>
<Authorization_File>
   <AuthorizedGroup ID="ngop_admin">
         <User Name="user_name"/>
         ...
   </AuthorizedGroup>
</Authorization_File>
</NGOPAction>
```

An `<Authorization_File>` tag contains zero or more `<AuthorizedGroup>` tags. These tags have a required attribute of `ID` and contain zero or more user names (`<User>`). This configuration file should conform to the DTD rules.

Example:
Two groups (`ngop_admin` and `oss_admin`) are described in this example. A list of authorized users is attached to each group.

```
<AuthorizationFile>
<AuthorizedGroup ID="ngop_admin">
               <User Name="smith"/>
               <User Name="jones"/>
        </AuthorizedGroup>
        <AuthorizedGroup ID="oss_admin">
               <User Name="brown"/>
               <User Name="johnson"/>
</AuthorizedGroup>
</AuthorizationFile>
```

15.4 File `action.xml`

The `action.xml` configuration file describes actions which consist of executables or scripts, a host where they are located, and the groups that are authorized to perform this action. The `action.xml` file requires the following declaration and tags:

```
<?xml version='1.0'?>
<!DOCTYPE NGOPAction System “action.dtd”>
<NGOPAction>
<Action_File>
<NGOPAction>
<Action_File>
   <Action ID="action_name">
      <Host Name="host_name">
      <AuthorizedGroup ID="group_name"/>
        ...
      <Exec Path="command_name"/>
       ...
      </Host>
   </Action>
</Action_File>
```

An `<Action>` tag has one required attribute `(ID)` and contains several `<Host>` tags. A `<Host>` tag has a `Name` attribute and contains one or more `<AuthorizedGroup>` tags (with an `ID` attribute) and `<Exec>` tags (with a `Path` attribute). This configuration file should conform to the DTD rules.

Example:

Two actions are defined in this example. The first action allows `operator` and `oss_admin` groups to send email via an Action Server running on the host `ndem`..

```
<ActionFile>
<Action ID=”email”>
        <Host Name=”ndem”>
                <AuthorizedGroup ID=”operator”/>
                <AuthorizedGroup ID=”oss_admin”/>
                <Exec Path=”scripts/email”>
        </Host>
</Action>
</ActionFile>
```

Chapter 16: Controlling the NGOP Daemons

The NGOP package requires multiple processes to be running on multiple systems:

- The NCS, Broker, and Action Server on a central service machine.
- Ping agents on some machines which watch over other machines.
- Monitoring agents local to various systems.

To facilitate this, NGOP provides a simple mechanism for an administrator to write down what NGOP processes should be running on a given system, and to start them, stop them, and make sure that they are still running. This mechanism is also integrated with the UPS packaging system which has an umbrella mechanism to start processes needed for various UPS products at system startup.

16.1: The `/var/ngop` Directory.

The start/stop mechanism by default uses a directory tree under `/var/ngop` on each system to record what processes should be running on that system. The location of this directory can be changed by setting the environment variable `NGOP_START_DIR.`

As an example, suppose that you wanted to have two `swatch_agent` processes each running a different configuration file, and one `plugins_agent` process:

```
ngop swatch_agent –c cfg1.xml
ngop swatch_agent –c cfg2.xml
ngop plugins_agent –c cfg3.xml
```

To configure this you would place the configuration files under `/var/ngop` as follows:

```
/var/ngop/swatch_agent/cfg1.xml
/var/ngop/swatch_agent/cfg2.xml
/var/ngop/plugins_agent/cfg3.xml
```

The directory tree should have `r+w` permissions for the `uid` who will be running the NGOP processes.

16.2 Starting the Agents

Once the directory structure has been setup under `/var/ngop`, the agents are started by doing one of the following:

```
ups start ngop
```

or

```
setup ngop
ngop start
```

When the command is issued, an informational message will be displayed to the screen for each process started. The start/stop mechanism also records which agents have been started with their process ID numbers in `/var/ngop/.pids.<hostname>.`

16.3 Monitoring the Agents

The start/stop mechanism provides a means to monitor the agents:

```
ups status ngop
```

or

```
setup ngop

ngop status
```

Below is an example of the output produced by the `status` command:

```
$ ngop status
```

```
Running:
PID     COMMAND
9707    ngop swatch_agent –c /var/ngop/swatch_agent/cfg1.xml
9710    ngop swatch_agent –c /var/ngop/swatch_agent/cfg2.xml
9713    ngop plugins_agent –c /var/ngop/plugins_agent/cfg3.xml
```

If one of the agents has died, it will still show in the listing:

```
$ ngop status
Running:
PID     COMMAND
9707    ngop swatch_agent –c /var/ngop/swatch_agent/cfg1.xml
died    ngop swatch_agent –c /var/ngop/swatch_agent/cfg2.xml
9713    ngop plugins_agent –c /var/ngop/plugins_agent/cfg3.xml
```

There is also a check command which will obtain the status of the agents and restart those that have died:

```
$ ngop check
Running:
PID     COMMAND
9707    ngop swatch_agent –c /var/ngop/swatch_agent/cfg1.xml   still running
9710    ngop swatch_agent –c /var/ngop/swatch_agent/cfg2.xml   died, restarting ...
9713    ngop plugins_agent –c /var/ngop/plugins_agent/cfg3.xml still running
```

16.4 Stopping the Agents

Once the directory structure under `/var/ngop` has been setup and the agents have been started as in the previous section, stopping the agents is accomplished with one of the following commands:

```
ups stop ngop
```

or

```
setup ngop
ngop stop
```

The stopping mechanism looks under `/var/ngop/.pids.<hostname>` for agents that have been started. Informational messages are displayed as each agent is stopped:

```
$ngop stop
Stopping: ngop swatch_agent –c /var/ngop/swatch_agent/cfg1.xml
Stopping: ngop swatch_agent –c /var/ngop/swatch_agent/cfg2.xml
Stopping: ngop plugins_agent –c /var/ngop/plugins_agent/cfg2.xml
```

16.5 Disabling/Enabling Agents

It is sometimes desirable to disable an agent, but not to discard it's configuration. This is accomplished with the `ngop disable` command. The agent is enabled with `ngop enable`. The argument to these commands is either the full NGOP command (as listed by `ngop status`) in quotes, or the base name of the configuration file (cfg2 for example):

```
$ ngop status
Running:
PID     COMMAND
9707    ngop swatch_agent –c /var/ngop/swatch_agent/cfg1.xml
9710    ngop swatch_agent –c /var/ngop/swatch_agent/cfg2.xml
9713    ngop plugins_agent –c /var/ngop/plugins_agent/cfg3.xml

$ ngop disable “cfg2”
Stopping: ngop swatch_agent –c /var/ngop/swatch_agent/cfg2.xml

$ngop stop
```

```
Stopping: ngop swatch_agent –c /var/ngop/swatch_agent/cfg1.xml
Stopping: ngop plugins_agent –c /var/ngop/plugins_agent/cfg2.xml

$ngop start
Starting: ngop swatch_agent –c /var/ngop/swatch_agent/cfg1.xml
Disabled: ngop swatch_agent –c /var/ngop/swatch_agent/cfg2.xml
Starting: ngop plugins_agent –c /var/ngop/swatch_agent/cfg3.xml

$ngop enable “cfg2”
Starting: ngop swatch_agent –c /var/ngop/swatch_agent/cfg2.xml
```

16.6 Controlling Agents on Remote Hosts

NGOP agents often run on multiple hosts. The `ngop remote` command is used to stop, start, or modify the behaviour of NGOP agents on remote hosts provided that the user issuing the command has permission to `rsh` to those hosts.

The format of the `remote` command is:

```
        ngop remote [-l user] <host>|<cluster> |
        ngop remote [-l user] <host_prefix>:<start_range>-<end_range>
```

For example, to start nodes `fcdf09`, `fcdf10`, `fcdf11`, `fcdf12`, and `fcdf13`, the following command is used:

```
        ngop remote fcdf:09-13
```

If the above command should be executed as the user `ngopuser`, the following command would be used:

```
        ngop remote –l ngopuser fcdf:09-13
```

If the “tictac” tools for the farms are being used, the tictac cluster name can be used:

```
        ngop remote –l ngop start –c fcdf_cluster
```

16.7 Starting/Stopping Individual Agents

The `start` and `stop` commands can be given a string to match that will pick servers to start or stop. For example, to stop a `swatch_agent` using the configuration file `/var/ngop/swatch_agent/cfg1.xml`:

```
$ngop stop “cfg1”

Stopping: ngop swatch_agent –c /var/ngop/swatch_agent/cfg1.xml
```

```
$ngop status

Running:

PID     COMMAND
died    ngop swatch_agent –c /var/ngop/swatch_agent/cfg1.xml
9708    ngop swatch_agent –c /var/ngop/swatch_agent/cfg2.xml
9713    ngop plugins_agent –c /var/ngop/plugins_agent/cfg3.xml
```

With multiple hosts, this can result in a more complicated string. For example, to kill the deamon in the above example on hosts `fcdf09-13`, you would issue the following command:

```
ngop remote ‘stop cfg1’ fcdf09-13
```

Appendix A

<For> DTD

```
<!ELEMENT  For ( #PCDATA| For* ) >
<!ATTLIST  For
       Each  CDATA #REQUIRED
       In    CDATA #REQUIRED
       Name  CDATA #REQUIRED
       Var   CDATA #REQUIRED
       File  CDATA #IMPLIED
>
```

<Apply> DTD

```
<!ELEMENT  apply  ( ( sum | min | max | divide | times | plus | minus | and | or | eq |
neq | gt | geq | lt | leq | in | notin ), ( apply | cn | ci )* ) >

<!ELEMENT  sum ( bvar, uplimit, lowlimit, ( apply | ci | cn )* ) >
<!ELEMENT  min ( bvar, uplimit, lowlimit, ( apply | ci | cn )* ) >
<!ELEMENT  max ( bvar, uplimit, lowlimit, ( apply | ci | cn )* ) >
<!ELEMENT  bvar EMPTY >
<!ELEMENT  uplimit ( apply | cn ) >
<!ELEMENT  lowlimit ( apply | cn ) >
<!ELEMENT  divide EMPTY >
<!ELEMENT  times EMPTY >
<!ELEMENT  plus EMPTY >
<!ELEMENT  minus EMPTY >
<!ELEMENT  and EMPTY >
<!ELEMENT  or EMPTY >
<!ELEMENT  eq EMPTY >
<!ELEMENT  neq EMPTY >
<!ELEMENT  gt EMPTY >
<!ELEMENT  geq EMPTY >
<!ELEMENT  lt EMPTY >
<!ELEMENT  leq EMPTY >
<!ELEMENT  in EMPTY >
<!ELEMENT  notin EMPTY >
<!ELEMENT  ci ( #PCDATA ) >
<!ELEMENT  cn ( #PCDATA ) >
```

<Action> DTD

```
<!ELEMENT  Action  ( Exec )+  >
<!ATTLIST  Action
       ID CDATA #REQUIRED
       Host  CDATA #REQUIRED
       Method ( manual | automatic ) 'automatic'
       Type  ( local | central ) 'central'
       Counter  CDATA #IMPLIED
       Gap  CDATA #IMPLIED
       Delay CDATA #IMPLIED
>
<!ELEMENT  Exec  EMPTY >
<!ATTLIST  Exec
       Name  CDATA #REQUIRED
       Argument  CDATA #REQUIRED
>
```

<If> DTD

```
<!ELEMENT  If ( #PCDATA|Else?) >
```

```
<!ATTLIST  If
       Cond  CDATA #REQUIRED “’{%Role}’==(!=)’role_name’”
>
<!ELEMENT Else (#PCDATA)>
```

NCS Configuration File DTD

```
<!ELEMENT  NCS_cfg  ( NCS,Client?,Agent)>
<!ATTLIST  NCS_cfg
        DebugLevel  CDATA #IMPLIED
>
<!ELEMENT  NCS  EMPTY >
<!ATTLIST  NCS
        TcpPort  CDATA #REQUIRED
        UdpPort  CDATA #REQUIRED
>
<!ELEMENT  Client  EMPTY >
<!ATTLIST  Client
        Name  "Archiver" #IMPLIED
        Port  CDATA #REQUIRED
        LocalLog  CDATA “log.log”
        Host  CDATA “localhost”
>

<!ELEMENT  TrustedDomain (Domain)+>
<!ELEMENT Domain Empty>
<!ATTLIST Domain
       Name CDATA #REQUIRED
>

<!ELEMENT  Agent  ( Action ?) >
<!ATTLIST  Agent
        UpdateInt  CDATA “2”
        TotalMsgNum  CDATA “400”
        TotalMsgLength  CDATA “100000”
        Window  CDATA “5”
         MissedHeartbeat “3”
>
<!ELEMENT  Action  ( Exec ) >
<!ATTLIST  Action
        ID  CDATA #REQUIRED
        Host  CDATA #IMPLIED
>
<!ELEMENT  Exec  EMPTY >
<!ATTLIST  Exec
        Argument  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
```

Locator Server DTD

```
<!-- Locator Server definitions starts -->
<!ELEMENT  LS_cfg  ( LS ) >
<!ATTLIST  LS_cfg
        DebugLevel CDATA "1"
>
<!-- Debug Level from 0 to 3 -->
<!ELEMENT  LS  EMPTY >
<!ATTLIST  LS
        InitWait CDATA "120"
        MCPort CDATA "3111"
        SEPort CDATA "20000"
>
<!-- time in seconds Locator Server will wait on startup for Status Engines  to register -
->
<!-- MCPort opened for Monitoirng Client connections -->
<!-- SEPort opened for Status Engine Connections  -->
```

```
<!-- Ports starting from 70001 will be allocated for Status Engines to open connection
with Monitoring Clients -->
<!-- Locator Server definitions ends -->
```

Status Engine Configuration File DTD

```
<!-- Status Engine definitions starts -->
<!ELEMENT  status_engine_cfg ( (Client)+,(CfgXml | ColorMap |  CfgEvnt | IconMap ))>
<!ATTLIST  status_engine_cfg
        DebugLevel  CDATA  "1"
>
<!-- Debug Level from 0 to 6 -->
<!ELEMENT  Client  EMPTY >
<!ATTLIST  Client
        port  CDATA #REQUIRED
        host  CDATA #REQUIRED
        name  (LSClnt|NCSClnt|CFMSClnt) #REQUIRED
>
<!-- Port and host for Locator Server, Central Server, and CFMS -->
<!ELEMENT  CfgXml  EMPTY >
<!ATTLIST  CfgXml
        cvsRep  CDATA #REQUIRED
        wrkDir  CDATA #REQUIRED
        cvsRoot  CDATA #IMPLIED
        role  CDATA #REQUIRED
        cfgRoot  CDATA #REQUIRED
>
<!-- Location definition for configuration files-->
<!-- cvsRep either name of cvs repository  or root directory for all the configuration
files -->
<!-- wrkDir parent directory for cvsRep -->
<!-- cvsRoot CVSROOT if cvs is in use -->
<!-- status engine role -->
<!-- cfgRoot - name of the root monitored object -->

<!ELEMENT  CfgEvnt  EMPTY >
<!ATTLIST  CfgEvnt
        Mail CDATA #IMPLIED
        EventRetentionInt  CDATA "24"
        WeekendRetentionInt CDATA "72"
        WeekendDay  (Sat|Mon|Tue|Wed|Thu|Fri|Sat) "Fri"
        WeekendStartTime
(0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24) "17"

>
<!-- EventRetentionInt: for how long events and alarms (hours) should be kept in memory --
>
<!-- Weekend definition: for how long events and alarms should be kept during weekend and
when weekend starts -->

<!ELEMENT  IconMap  ( Type )* >
<!ELEMENT  Type  EMPTY >
<!ATTLIST  Type
        Icon  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>

<!ELEMENT  ColorMap  ( Status )* >
<!ELEMENT  Status  EMPTY >
<!ATTLIST  Status
        Name  CDATA #REQUIRED
        Color  CDATA #REQUIRED
>
```

PlugIns Agent DTD

```
<!ELEMENT  MA-config  (NCS,( System  )*) >
<!ATTLIST  MA-config
        Update  CDATA "180"
        Name  CDATA #REQUIRED
        Type (Cron|Daemon) "Daemon"
>
<!ELEMENT  NCS  EMPTY >
<!ATTLIST  NCS
        Port  CDATA "19997"
        Host  CDATA #REQUIRED
        Heartbeat  CDATA "600"
>
<!ELEMENT  System  ( (ConditionSet)* , (MonitoredElement)* ) >
<!ATTLIST  System
        Cluster  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!ELEMENT  MonitoredElement  ( ConditionSet )* >
<!ATTLIST  MonitoredElement
        Name  CDATA #REQUIRED
        Type  CDATA #REQUIRED
        Host  CDATA #REQUIRED
>

<!ELEMENT  ConditionSet  ( fn, (Condition)+) >
<!ELEMENT  fn  EMPTY >
<!ATTLIST  fn
        Name  CDATA "plug_ins"
        Arg  CDATA #REQUIRED
        RetVal  CDATA #REQUIRED
>
<!ELEMENT  Condition  ( (apply)+, ( Action )*) >
<!ATTLIST  Condition
        Description  CDATA #REQUIRED
        SevLevel  CDATA #REQUIRED
        State  CDATA #REQUIRED
        EventType  CDATA #IMPLIED
        EventName  CDATA #IMPLIED
>
<!ELEMENT  Action  ( Exec )* >
<!ATTLIST  Action
        ID  CDATA #REQUIRED
        Type  CDATA #REQUIRED
        Host  CDATA #REQUIRED
        Gap  CDATA #IMPLIED
        Counter  CDATA #IMPLIED
        Delay CDATA #IMPLIED
>
<!ELEMENT  Exec  EMPTY >
<!ATTLIST  Exec
        Argument  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!-- see for dtd -->
<!-- see apply dtd -->
<!-- see action dtd -->
```

Ping Agent DTD

```
<!ELEMENT  MA-config  (NCS, (DefaultFiles) ?( System  )*) >
<!ATTLIST  MA-config
        Update  CDATA "180"
        Name  CDATA #REQUIRED
        Type (Cron|Daemon) "Daemon"
>
<!ELEMENT  NCS  EMPTY >
<!ATTLIST  NCS
        Port  CDATA "19997"
        Host  CDATA #REQUIRED
        Heartbeat  CDATA "600"
>
<!ELEMENT  DefaultFiles (File)* >
<!ATTLIST  DefaultFiles
        Type ( "KnownStatus” | HostsInClusters” ) #REQUIRED
        Path  CDATA #REQUIRED
>
<!ELEMENT  File Empty >
<!ATTLIST  File
        Name CDATA #REQUIRED
>

<!ELEMENT  System  ( (ConditionSet)* , (MonitoredElement)* ) >
<!ATTLIST  System
        Cluster  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!ELEMENT  MonitoredElement  ( ConditionSet )* >
<!ATTLIST  MonitoredElement
        Name  CDATA #REQUIRED
        Type  CDATA #REQUIRED
        Host  CDATA #REQUIRED
>

<!ELEMENT  ConditionSet  ( fn, (Condition)+) >
<!ELEMENT  fn  EMPTY >
<!ATTLIST  fn
        Name  (isNetworkDown | isUp | getLoad ) #REQUIRED
        Arg  CDATA #REQUIRED
        RetVal  CDATA #REQUIRED
>
<!ELEMENT  Condition  ( (apply)+, ( Action )*) >
<!ATTLIST  Condition
        Description  CDATA #REQUIRED
        SevLevel  CDATA #REQUIRED
        State  CDATA #REQUIRED
        EventType  CDATA #IMPLIED
        EventName  CDATA #IMPLIED
>
<!ELEMENT  Action  ( Exec )* >
<!ATTLIST  Action
        ID  CDATA #REQUIRED
        Type  CDATA #REQUIRED
        Host  CDATA #REQUIRED
        Gap  CDATA #IMPLIED
        Counter  CDATA #IMPLIED
        Delay CDATA #IMPLIED
>
<!ELEMENT  Exec  EMPTY >
<!ATTLIST  Exec
        Argument  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!-- see for dtd -->
<!-- see apply dtd -->
<!-- see action dtd -->
```

Swatch Agent DTD

```
<!ELEMENT  SwatchAgentConfig  ( NCS ,(File | If_File | For_File | Action)*) >
<!ATTLIST  SwatchAgentConfig
        Scan  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!ELEMENT  File  ( System | For_System | If_System )* >
<!ATTLIST  File
        Filetype (multihost|plain) "plain"
        File  CDATA #REQUIRED
>
<!ELEMENT  MonitoredElement  ( ReRule|If_ReRule|For_ReRule )* >
<!ATTLIST  MonitoredElement
        Type  CDATA #REQUIRED
        Host  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!ELEMENT  Action  ( Exec )* >
<!ATTLIST  Action
        Type  CDATA #REQUIRED
        ID  CDATA #REQUIRED
        Local  CDATA #REQUIRED
>
<!ELEMENT  ReRule  EMPTY >
<!ATTLIST  ReRule
        Regexp  CDATA #REQUIRED
        State  CDATA #REQUIRED
        SevLevel  CDATA #REQUIRED
        EventValue  CDATA #IMPLIED
        EventName  CDATA #REQUIRED
        EventType  CDATA #REQUIRED
        Description  CDATA #IMPLIED
>
<!ELEMENT  System  ( MonitoredElement | If_MonitoredElement | For_MonitoredElement )* >
<!ATTLIST  System
        Cluster  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!ELEMENT  NCS EMPTY >
<!ATTLIST  NCS
        Port  CDATA #REQUIRED
        Host  CDATA #REQUIRED
        Heartbeat  CDATA #REQUIRED
>
<!ELEMENT  Exec  EMPTY >
<!ATTLIST  Exec
        Argument  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!ELEMENT  For_System  ( If_System|For_System|System )* >
<!ATTLIST  For_System
        Each  CDATA #REQUIRED
        In  CDATA #REQUIRED
        Name  CDATA #REQUIRED
        Filename  CDATA #IMPLIED
        Var  CDATA #REQUIRED
>
<!ELEMENT  If_System  ( If_System|For_System|System )* >
<!ATTLIST  If_System
        Cond  CDATA #REQUIRED
>
<!ELEMENT  For_MonitoredElement  (
If_MonitoredElement|For_MonitoredElement|MonitoredElement )* >
<!ATTLIST  For_MonitoredElement
        Each  CDATA #REQUIRED
        In  CDATA #REQUIRED
        Name  CDATA #REQUIRED
        Filename  CDATA #IMPLIED
        Var  CDATA #REQUIRED
>
<!ELEMENT  If_MonitoredElement  (
If_MonitoredElement|For_MonitoredElement|MonitoredElement )* >
<!ATTLIST  If_MonitoredElement
        Cond  CDATA #REQUIRED
>
<!ELEMENT  For_File  ( If_File|For_File|File )* >
<!ATTLIST  For_File
        Each  CDATA #REQUIRED
        In  CDATA #REQUIRED
        Name  CDATA #REQUIRED
        Filename  CDATA #IMPLIED
        Var  CDATA #REQUIRED
>
```

```
<!ELEMENT  If_File  ( If_File|For_File|File )* >
<!ATTLIST  If_File
        Cond  CDATA #REQUIRED
>
<!ELEMENT  For_ReRule  ( IfReRule|For_ReRule|ReRule )* >
<!ATTLIST  For_ReRule
        Each  CDATA #REQUIRED
        In  CDATA #REQUIRED
        Name  CDATA #REQUIRED
        Filename  CDATA #IMPLIED
        Var  CDATA #REQUIRED
>
<!ELEMENT  If_ReRule  ( If_ReRule|For_ReRule|ReRule )* >
<!ATTLIST  If_ReRule
        Cond  CDATA #REQUIRED
>
```

URL Agent DTD

```
<!ELEMENT  URLAgentConfig (NCS, (If_System|For_System|System|Action)+) >
<!ATTLIST  URLAgentConfig
        Scan  CDATA #REQUIRED
        name  CDATA #REQUIRED
>
<!ELEMENT  NCS  EMPTY >
<!ATTLIST  NCS
        Port  CDATA #REQUIRED
        Host  CDATA #REQUIRED
        Heartbeat  CDATA #REQUIRED
>
<!ELEMENT  System  ( If_MonitoredElement|For_MonitoredElement|MonitoredElement )+ >
<!ATTLIST  System
        Cluster  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!ELEMENT  MonitoredElement  ( NGOP_URL | If_URLFailRule | For_URLFailRule | URLFailRule
)* >
<!ATTLIST  MonitoredElement
        Name  CDATA #REQUIRED
        Type  CDATA #REQUIRED
        Host  CDATA #REQUIRED
>

<!ELEMENT  URLFailRule  EMPTY >
<!ATTLIST  URLFailRule
        href  CDATA #REQUIRED
        RegExp  CDATA #REQUIRED
        ActionLocal  CDATA #IMPLIED
>
<!ELEMENT  NGOP_URL  EMPTY >
<!ATTLIST  NGOP_URL
        ActionLocal  CDATA #IMPLIED
>
<!ELEMENT  Action  ( Exec )* >
<!ATTLIST  Action
        Type  CDATA #REQUIRED
        ID  CDATA #REQUIRED
        Local  CDATA #REQUIRED
>
<!ELEMENT  Exec  EMPTY >
<!ATTLIST  Exec
        Argument  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!ELEMENT  For_System  ( If_System|For_System|System )* >
<!ATTLIST  For_System
        Each  CDATA #REQUIRED
        In  CDATA #REQUIRED
        Name  CDATA #REQUIRED
        Filename  CDATA #IMPLIED
        Var  CDATA #REQUIRED
>
<!ELEMENT  If_System  ( If_System|For_System|System )* >
<!ATTLIST  If_System
        Cond  CDATA #REQUIRED
>
<!ELEMENT  For_MonitoredElement  (
If_MonitoredElement|For_MonitoredElement|MonitoredElement )* >
<!ATTLIST  For_MonitoredElement
        Each  CDATA #REQUIRED
        In  CDATA #REQUIRED
        Name  CDATA #REQUIRED
        Filename  CDATA #IMPLIED
        Var  CDATA #REQUIRED
>
<!ELEMENT  If_MonitoredElement  (
If_MonitoredElement|For_MonitoredElement|MonitoredElement )* >
<!ATTLIST  If_MonitoredElement
        Cond  CDATA #REQUIRED
>
<!ELEMENT  For_URLFailRule  ( If_URLFailRule|For_URLFailRule|URLFailRule )* >
<!ATTLIST  For_URLFailRule
        Each  CDATA #REQUIRED
        In  CDATA #REQUIRED
        Name  CDATA #REQUIRED
        Filename  CDATA #IMPLIED
        Var  CDATA #REQUIRED
>
<!ELEMENT  If_URLFailRule  ( If_URLFailRule|For_URLFailRule|URLFailRule )* >
<!ATTLIST  If_URLFailRule
        Cond  CDATA #REQUIRED
>
```

<Default_File> DTD

```
<!ELEMENT  NGOPConfig  (Default_File, (HostsInClusters|KnownStatus|ServiceClass )) >
<!ELEMENT  Default_File  EMPTY >
<!ELEMENT  If  (#PCDATA|Else)* >
<!ATTLIST  If
        Cond  CDATA #REQUIRED
>
<!ELEMENT  Else (#PCDATA)* >
<!ELEMENT  For  (#PCDATA)* >
<!ATTLIST  For
        Each  CDATA #REQUIRED
        In  CDATA #REQUIRED
        Name  CDATA #REQUIRED
        Var  CDATA #REQUIRED
        Filename  CDATA #IMPLIED
>

<!ELEMENT  HostsInClusters  ( Cluster )+ >
<!ELEMENT  ServiceType  ( Host*|apply+ ) >
<!ATTLIST  ServiceType
        Name  (24by7| 8to17by5 |8to17by7|8to00by7|0by0) #REQUIRED
>
<!ELEMENT  ServiceClass  ( ServiceType )* >
<!ELEMENT  KnownStatus  ( Status )* >
<!ELEMENT  Status  ( OutOfServiceInterval )* >
<!ATTLIST  Status
        Name  (bad|test|in_repair)  #REQUIRED
>
<!ELEMENT  OutOfServiceInterval  ( System | MonitoredElement | Host | Cluster )+ >
<!ATTLIST  OutOfServiceInterval
        StartDateTime CDATA "None"
        EndDateTime CDATA "None"
        User  CDATA #IMPLIED
        Description  CDATA #IMPLIED
        Cron CDATA #IMPLIED
>
<!ELEMENT  Host  EMPTY >
<!ATTLIST  Host
        Name  ID #REQUIRED
>
<!ELEMENT  Cluster  ( ServiceType | Cluster )* >
<!ATTLIST  Cluster
        Name  ID  #REQUIRED
>
<!ELEMENT  System  EMPTY>
<!ATTLIST  System
        Cluster  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!ELEMENT  MonitoredElement  EMPTY >
<!ATTLIST  MonitoredElement
        System  CDATA #REQUIRED
        Host  CDATA #REQUIRED
        Cluster  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!-see apply dtd --!>
<!-see for dtd --!>
```

Monitored Hierarchy DTD

```
<!ELEMENT  NGOPHierarchy  (SystemView|System)* >
<!ELEMENT  SystemView  (SystemView|System)+>
<!ATTLIST  SystemiView
        Name  ID #REQUIRED
        RefRule CDATA #IMPLIED
>
<!ELEMENT  System  (MonitoredElement)+>
<!ATTLIST  System
        Cluster  CDATA #REQUIRED
        Name  CDATA #REQUIRED
        RefRule CDATA #IMPLIED
>
<!ELEMENT  MonitoredElement  EMPTY >
<!ATTLIST  MonitoredElement
        Host  CDATA #REQUIRED
        Name  CDATA #REQUIRED
        Type CDATA #IMPLIED
        RefRule CDATA #IMPLIED
>
```

<StatusRulesSet> DTD

```
<!ELEMENT  NGOPRule  ( StatusRulesSet )* >
<!ELEMENT  StatusRulesSet  ( DependList?, ( GenricRule | DependRule )* ) >
<!ATTLIST  StatusRulesSet
       ID  CDATA #REQUIRED
>
<!ELEMENT  DependList  (Group )* >
<!ELEMENT Group ( For | System )* >
<!ATTLIST  Name
       Name  CDATA #REQUIRED
>
<!-- see system dtd in hierarchy -->
<!ELEMENT  GenericRule  ( apply, Action ) >
<!ATTLIST  GenericRule
       Prio  CDATA #REQUIRED
       Status  ( None|Good|Unknown|Undefined|Warning|Error|Bad ) #REQUIRED
       SevLevel  ( None|Good|Unknown|Undefined|Warning|Error|Bad )  'Good'
       Dsc CDATA #IMPLIED
>

<!ELEMENT  DependRule  ( apply, Action ) >
<!ATTLIST  DependRule
       Prio  CDATA #REQUIRED
       Status  (None|Good|Unknown|Undefined|Warning|Error|Bad) #REQUIRED
       SevLevel  (None|Good|Unknown|Undefined|Warning|Error|Bad)  'Good'
       Dsc CDATA #IMPLIED
>
<!--see for dtd -->
<!--see apply dtd -->
<!--see action dtd -->
```

Web Gui DTD

```
<!ELEMENT  webmonitor_cfg  ( LS , WebGui ) >
<!ELEMENT  LS  EMPTY >
<!ATTLIST  LS
        Host  CDATA #REQUIRED
        Port  CDATA #REQUIRED
>
<!ELEMENT  WebGui  EMPTY >
<!ATTLIST  WebGui
        Type  (-fcgi | -cgi |-standalone)
>
```

Java Monitor DTD

```
<!ELEMENT jmonitor_cfg (ls,roles)>
<!ELEMENT ls  EMPTY >
<!ATTLIST ls
        Port  CDATA #REQUIRED
        Host  CDATA #REQUIRED
>
```

CFMS Configuration File DTD

```
<!ELEMENT  client_cfg  ( Client | CfgXml ) >
<!ELEMENT  Client  EMPTY >
<!ATTLIST  Client
        Host  CDATA #REQUIRED
        Port  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!ELEMENT  CfgXml  EMPTY >
<!ATTLIST  CfgXml
        CvsRoot  CDATA #REQUIRED
        Name  CDATA #REQUIRED
        Version  CDATA #REQUIRED
        CvsRep  CDATA #REQUIRED
>
```

Archiver Configuration File DTD

```
<!ELEMENT  ArchiverConfig  ( Port, ArchiverHost, LogPath, RequestDirectory,
ErrorDirectory, DBInterSleepInterval ) >
<!ELEMENT  Port  EMPTY >
<!ELEMENT  ArchiverHost  ( OraPW, OraUser, OraInstance ) #REQUIRED >
<!ELEMENT  LogPath  EMPTY >
<!ELEMENT  RequestDirectory  EMPTY >
<!ELEMENT  ErrorDirectory  EMPTY >
<!ELEMENT  DBInterSleepInterval  EMPTY >
<!ELEMENT  OraPW  EMPTY >
<!ELEMENT  OraUser  EMPTY >
<!ELEMENT  OraInstance  EMPTY >
```

Action Server Configuration File DTD

```
<!-- Action Server definitions starts -->
<!ELEMENT AS_cfg  ( Client |CfgXml | ActionObjectList)+ >
<!ATTLIST  AS_cfg
       DebugLevel CDATA "1"
>
<!-- Debug Level from 0 to 6 -->
<!ELEMENT  Client  EMPTY >
<!ATTLIST  Client
        Port  CDATA "19996"
        Name  (NCSClnt|CFMSClnt) "NCSClnt"
        Host  CDATA "localhost"
>
<!-- NCSClnt Client connects to NCS -->
<!-- CFMSClnt connects to CFMS  -->
<!ELEMENT  CfgXml  EMPTY >
<!ATTLIST  CfgXml
        ExcDir  CDATA "scripts"
        WrkDir  CDATA ".ngop_action"
        CvsRep  CDATA "configxml"
        CvsRoot CDATA ":pserver:ngop@ngop.fnal.gov:/home/ngop/Repository"
        Role    CDATA "default"
>
<!ELEMENT ActionObjectList (MonitoredElement,System,Cluster,Host)* >
<!ELEMENT  Host  EMPTY >
<!ATTLIST  Host
        Name  ID #REQUIRED
>
<!ELEMENT  Cluster EMPTY >
<!ATTLIST  Cluster
        Name  ID  #REQUIRED
>
<!ELEMENT  System  EMPTY>
<!ATTLIST  System
        Cluster  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!ELEMENT  MonitoredElement  EMPTY >
<!ATTLIST  MonitoredElement
        System  CDATA #REQUIRED
        Host  CDATA #REQUIRED
        Cluster  CDATA #REQUIRED
        Name  CDATA #REQUIRED
>
<!-- Action Server definitions ends --><!ELEMENT  AS_cfg  ( Client+, CfgXml ) >
```

<Authorization_File> DTD

```
<!ELEMENT  NGOPAction  ( Authorization_File ) >
<!ELEMENT  Authorization_File  ( AuthorizedGroup )* >
<!ELEMENT  AuthorizedGroup  ( User )* >
<!ATTLIST  AuthorizedGroup
        ID  CDATA #REQUIRED
>
```

```
<!ELEMENT  User  EMPTY >
<!ATTLIST  User
        Name  CDATA #REQUIRED
>
```

<Action_File> DTD

```
<!ELEMENT  NGOPAction ( Action_File ) >
<!ELEMENT  Action_File  ( Action )* >
<!ELEMENT  Action  ( Host )* >
<!ATTLIST  Action
       ID  CDATA #REQUIRED
>
<!ELEMENT  Host  ( AuthorizedGroup , Exec )* >
<!ATTLIST  Host
       Name  CDATA #REQUIRED
>
<!ELEMENT  Exec  EMPTY >
<!ATTLIST  Exec
       Path  CDATA #REQUIRED
>
<!ELEMENT  AuthorizedGroup  EMPTY >
<!ATTLIST  AuthorizedGroup
       ID  CDATA #REQUIRED
>
```